Personal recommendations for diabetes´ control

Final degree project
Bachelor degree in IT Engineering

Anton Kovalev

Director:
Mª Belén Díaz Agudo
Co-director:
Juan Antonio Recio García

Department of Software Engineering and Artificial Intelligence
Faculty of Computer Science
Complutense University of Madrid

September 2017

Personal recommendations for diabetes´ control

Project specification presented by
Anton Kovalev
for obtaining Bachelor Degree in
IT Engineering

Directed by
Mª Belén Díaz Agudo, Ph.D.
Co-directed by
Juan Antonio Recio García, Ph.D.

Department of Software Engineering and Artificial Intelligence
Faculty of Computer Science
Complutense University of Madrid

September 2017

Autorización

Anton Kovalev, alumno matriculado en el Grado de Ingeniería Informática, especialidad Tecnología de la Información, autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor, la memoria, el código y la documentación de la aplicación online, todo ello realizado durante el curso académico 2016-2017 bajo la dirección de Mª Belén Díaz Agudo, Ph.D. y Juan Antonio Recio García, Ph.D., profesores del Departamento de Ingeniería del Software e Inteligencia Artificial.

_____ *Anton Kovalev*

Acknowledgments

First, I would like to thank my project directors, Mª Belén y Juan Antonio- you were guiding me during the last 1.5 years (since we´ve started this project as an out-of-course activity) and you showed me a very interesting world of case-based reasoning. It´s very important that you were always ready to help me when I was confused, but at the same time you made me work really hard- it was an extraordinary experience.

Then, I would like to specially mention my friend Alberto- we were working on this project during almost 1 year and your ideas were very useful. Thank you for those first mockups you have made- your knowledge and experience in web developing served me a lot.

This project required an experience in different areas of Computer Science, some of them new for me- so I appreciate my friends dedicating me their time and explaining new concepts.

Moreover, of course, thanks to my family. Not just for animating and cheering me during this year- but in general. In 2010, you proposed me to study abroad - this decision changed my life a lot and this work is a result of the previous 7 years.

Summary

According to the latest statistics of World Health Organization [1], 422 million adults have diabetes- a metabolic disease that is characterised by high blood sugar levels over a long period. Unfortunately, it cannot be completely cured and it may have very dangerous consequences, if not treated on time. That is why it is very important to find out this disease as soon as possible and to change your style of life in an appropriate way.

In 2014 a group of Spanish professors from the Department of Computer Architecture and Automation (DACYA, Complutense University of Madrid, [2]) created "glUCModel" [3] - a special system for patients with diabetes. This application improves the communication and interaction between patients and doctor: patients can upload their medical/personal data and doctor has a possibility to consult this information and to have better control over patient records. However, three interesting functions were still not implemented: an e-learning course (a space for patients to consult information about the illness and to amplify their knowledge), a module for automatic generation of glucose levels model (based on history and personal characteristics) and a recommender system (based on recorded data and physician preferences).

The main goal of this project is to create a case-based reasoning [4] system that will be able to give personal recommendations to every person according to the historic data about all patients. As a result of implementing and adding new functionalities, a new version of an application, called "glUCModel 2.0", was created.

Keywords: glUCModel, case-based reasoning, diabetes, recommender, telemedicine, web applications

Resumen

Según los últimos datos de la Organización Mundial de la Salud [1], 422 millones de personas tienen diabetes- una enfermedad metabólica caracterizada por altos niveles de glucosa en sangre durante un periodo de tiempo prolongado. Por desgracia, no puede ser completamente curada y puede tener unas consecuencias muy graves al no tratarla al tiempo. Por tanto, es muy importante diagnosticar esta enfermedad lo antes posible y cambiar su modo de vivir de forma adecuada.

En el año 2014 el grupo de profesores del Departamento de Arquitectura de Computadores y Automática (DACyA, Universidad Complutense de Madrid) creó "glUCModel" [3]- un sistema especial para pacientes con diabetes. Esa aplicación mejora la comunicación e interacción entre pacientes y doctor: pacientes pueden subir sus datos médicos/personales y el doctor tiene la posibilidad de consultar esa información y mejorar el control de los registros de pacientes. Sin embargo, 3 funciones muy interesantes todavía no estaban implementadas: un curso virtual de aprendizaje (un espacio para pacientes donde podrían obtener la información y amplificar sus conocimientos), un módulo de generación automática de un modelo de nivel de glucosa (basado en historial y características personalizadas) y el recomendador (basado en datos registrados y acciones de médico)

El objetivo principal de este proyecto es crear un sistema de razonamiento basado en casos [4] que podría dar recomendaciones personalizadas a cada persona basándose en los datos guardados de todos los pacientes. Han sido implementadas y añadidas nuevas funcionalidades, con lo cuál ha sido creada la nueva versión de aplicación "glUCModel 2.0"

Palabras clave: glUCModel, razonamiento basado en casos, diabetes, recomendador, telemedicina, aplicaciones Web

Table of Contents

List of figures

Chapter 1: Introduction

(English version)

Diabetes [5] is a very well-known disease: for the first time, it was mentioned in Egyptian manuscripts approx. 1500 years BCE as “too great emptying of urine”. The term “diabetes” (or “to pass through”) was first used in Ancient Greece in 3rd century CE and Type 1/Type 2 diabetes were identified for the first time by the Indian physicians in 400-500 CE: type 1 associated with youth and type 2- with overweight. The effective treatment was discovered just in 1920-s when Canadians scientists isolated and purified insulin.

Nowadays 9.6% of Spanish population have diabetes- it means that almost 4.3 million people need to control their weight, measure a blood sugar level, follow a special diet, etc. glUCModel [3] gives a possibility to have all this data stored in database and to communicate with your doctor to receive recommendations.

As we are talking about an application used by a hospital, it may have thousands of records. Therefore, it is possible that your situation seems a lot to the case of another patient. In this case (and thanks to a CBR [4] system), you will receive a recommendation in a few seconds, without waiting for doctor’s answer. Of course, he will be able to modify an “advice” given by the system and/or propose his own solution.

(Spanish version)

La diabetes es una de las enfermedades más antiguas que se conocen- por primera vez fue mencionada en los manuscritos de Egipto aprox. 1500 a.C. como “salida excesiva de orina”. El término “diabetes” (o “correr a través”) fue utilizado por primera vez en Grecia Antigua en el siglo III y los tipos 1 y 2 eran destacados en los años 400- 500 por los médicos indios: tipo 1 asociado con la juventud y el tipo 2- con el sobrepeso. Un tratamiento eficaz fue descubierto solo

en los años 1920, cuando los científicos canadienses consiguieron aislar y depurar insulina.

Hoy en día un 9.6% de la población española tiene diabetes- eso significa que casi 4.3 millones de personas tienen que controlar su peso, medir el nivel de glucosa en la sangre, seguir una dieta especial, etc. glUCModel [3] *da la posibilidad de tener todos estos datos almacenados en la base de datos y de comunicar con tu doctor para obtener recomendaciones.*

Como estamos hablando de una aplicación utilizada por el hospital, puede tener miles de registros. Por tanto, es posible que tu situación parece mucho a un caso de otro paciente. En este caso (y gracias al sistema CBR [4]*) vas a obtener la recomendación en unos segundos, sin esperar la respuesta del médico. Por supuesto, tendrá la posibilidad de modificar el "consejo" dado por el sistema y/o proponer su propia solución.*

Motivation

A patient with diabetes has to control many aspects of his life. He measures a blood sugar level at least once per day, regularly measures his weight and performs physical activity, makes insulin injections, passes other medical tests, etc.- all these actions generate a lot of data and a patient needs some space to store it. Moreover, digital glucometers, weighing scales and fitness machines have already been created, which means that records can be also automatically imported into the system. And of course, a patient needs to regularly communicate with a doctor and to receive an evaluation of a current situation, but sometimes there is no way to do it in person…

At the same time, in order to give a right recommendation, doctors should answer two questions. The first one is "How does the situation of a concrete patient change in time?"- and it means that a doctor should have access to previous records of patient´s medical history. The second one is "Was there a person with such symptoms earlier and, if yes, what did another doctor do?"

The main goal was to provide an access to data for both types of users and to facilitate the communication between them- and that´s "glUCModel" [3] was created for. Patients can easily manage records in a system, download reports of selected types and submitting data in two ways (manual or automatic[1]), doctors can see a visual representation of the most important parameters (dynamics of weight and blood sugar level) and, if necessary, go into details. But the most important thing is that today the evaluation can be performed automatically, basing on precedents and remotely- this is possible through the use of recommender, a key feature of "glUCModel 2.0". A patient can ask for advice by pressing one button- and a doctor, which may be located

[1] Automatic import is not available for all types of data- see Chapter 4 and 5

even in other country, will give him a personalized recommendation basing on patient´s profile, his own opinion and experience of other professionals.

General objective

The main objective is to create a Web application that can be used by doctors and patients. A created system will facilitate the communication between them, will permit to store and update all the relevant data and to generate personal recommendations, based on information about a patient and on similar cases of other patients.

Objectives:

1. To study the original version of "glUCModel" [3] to understand, what kind of functions I'll have to implement in my application;

2. To create a new database (it will be based on an old one, but some tables will be eliminated or modified);

3. To create a new Web interface that will be user-friendly and adaptable to different types of devices;

4. To create a Hibernate connection to the databases and to provide stable interchange of data between the "view" (web pages) and "model" (database);

5. To create a recommender (using CBR) and to add it to the system

Structure of this document

This document will be organized as described below:

- **Chapter 1: Introduction**
  In this Chapter I talk about the motivation of this project, about the general and objectives and I give this brief description of the document;

- **Chapter 2: Brief description of "glUCModel"**
  Here I talk about the first prototype of the system, mentioning important functions and problems (possible adjustments) of "glUCModel"- this is a starting point to make first mockups of a future system;

- **Chapter 3: Technologies and CBR State of the Art**
  This Chapter has two parts: first, there is a brief review of technologies that were used in this project and then we will talk

about theoretical footings of case-based reasoning applications and recommenders;

- **Chapter 4: Design of "glUCModel 2.0"**
  This is a detailed description of a functionality of a new application: requisites, description of users (modes of behavior), cases of use, implemented functions and limitations;

- **Chapter 5: Implementation of "glUCModel 2.0"**
  This part is dedicated to a detailed review of the database, contains examples of Web interfaces and a brief explication about the internal part of project (functionality of recommender, main classes and packets, etc.);

- **Conclusions**
  This Chapter contains main conclusions about the project: brief analysis of achievements, comparison of a result with expected goals and ideas of possible improvements

Chapter 2: Description of “glUCModel”

This section is dedicated to describing an original system. I will briefly analyse the first version of the application, trying to understand the main goal of creation, discovering basic functions and, of course, paying special attention to errors, mistakes and problems of the system- this will be the first and the most important step on my way to a new version.

First version of “glUCModel”

By the definition given in an article that was published in “Journal of Biomedical Information” [3], “glUCModel” is a “monitoring and modelling system for chronical diseases applied to diabetes”. This system was created, because chronic patients should strictly control their weight, diet, physical activity and blood glucose levels- however, it´s impossible to always have your doctor nearby.

The first part is a Web application, which gives to patients a possibility to upload their medical and personal data that will be stored on a server (in a database). The second part is a recommender- the “brain” of system, which will process the incoming data and, when a request will be received, will provide a personalized recommendation. These messages are supposed to improve their way of life and to give them additional information about diabetes. Finally, there is “Moodle”- an e-learning platform for patients, where they will find a detailed information about every aspect of diabetes and will be able to evaluate their knowledge by special tests.

In 2015, mobile applications for “glUCModel” were created. Both versions (for Android [*Figure 1*] [6] and for iOS [*Figure 2*] [7]) have a similar functionality: there are two modes of use (patient and doctor), as a patient, you may upload information about diet, glycaemia, injections of insulin, exercises and/or weight or access to your recommendations. If you are a doctor, you may see the list of your patients, register a new one, create a new recommendation and, if necessary, send it to a concrete person.

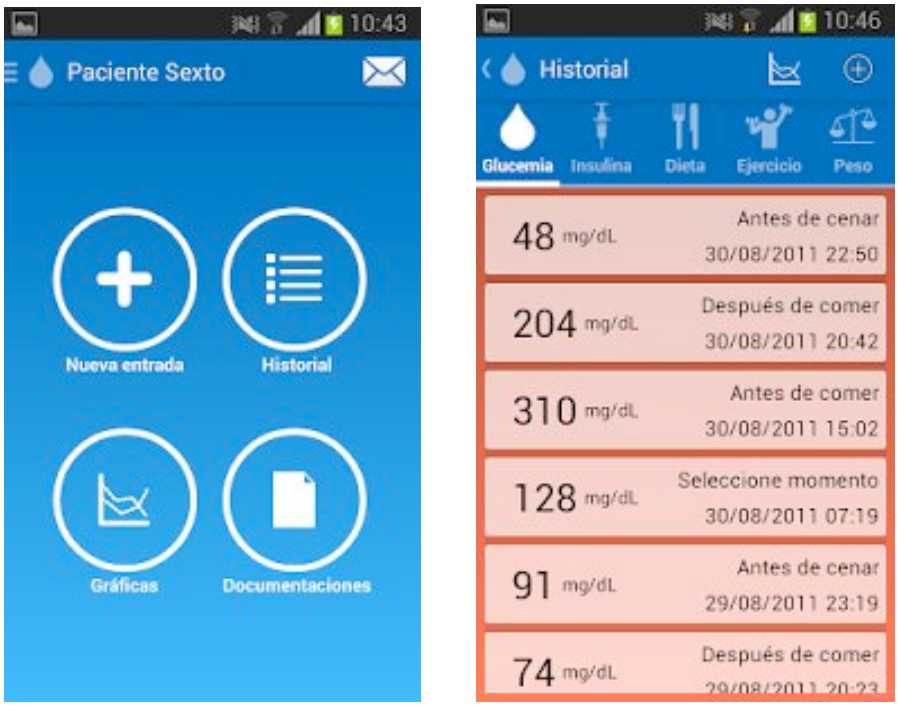

Figure 1: Android print screens

Figure 2: iOS print screens

Functionality of "glUCModel"

Patient options

As we mentioned above, there are two modes of use (patient and doctor). If a user is logged in like a patient, (s)he will see the next page [*Figure 3*]*:*

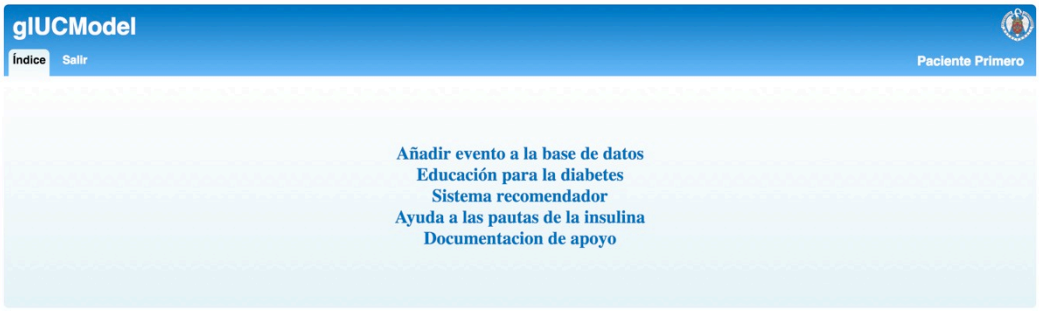

Figure 3: Old patient index

There are five main options in the menu:

1. **To add new information to the database.** There are six types of data in an application: glycemias, insulin injections, diet, exercises, weights and medical tests. All the records are grouped in tables- and, although the function is called "ADD information", user can list, modify or delete any record. He is also able to import and/or export data- but there is no possibility to choose a concrete type of records to download. In a new version, this function is implemented (data can be downloaded from a correspondent section);

2. **Education for diabetics.** This one is a link to a Moodle system: it is not implemented neither in this system nor in my project;

3. **Recommender system.** This section has two tabs: a "mailbox", where a patient can see received recommendations, and an "evaluate" button, that is supposed to send a request of evaluation to a doctor. In a new version, the patient will have just the first option and the format of received recommendations is simplified- for example, there is no sense to separate data and time of arrival of a recommendation, so in a new version it will be the same field;

4. **Advices about insulin**. This section was created to give recommendations about the use and injections of insulin to a patient. However, there is a link for this option, it was not implemented in a first version of "glUCModel". In "glUCModel 2.0" this option works as a medical reference book: a patient or a doctor can search for any medicine and download it´s data sheet or leaflet;

5. **Documentation.** Patient may have access to some documents that are created and submitted by doctors. This option is not implemented in my system

In general, a new application now has a similar functionality comparing to a present system- however, there are some new functions, such as an export of a concrete type of data for a concrete patient. Moreover, the design of a patient welcome page is changed: now it´s possible to see there a basic information about a person and to access directly to every type of data. Functions 2-4 from the previous list were transformed into buttons on a top of the screen [*Figure 4*]:

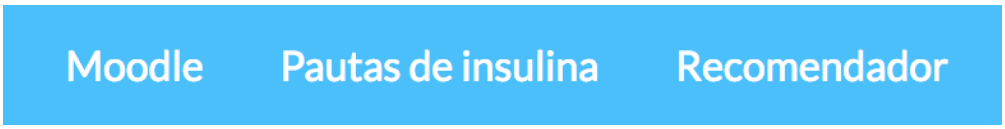

Figure 4: Patient´s functions

Doctor options

If the user is a doctor, a start page looks like this [*Figure 5*]:

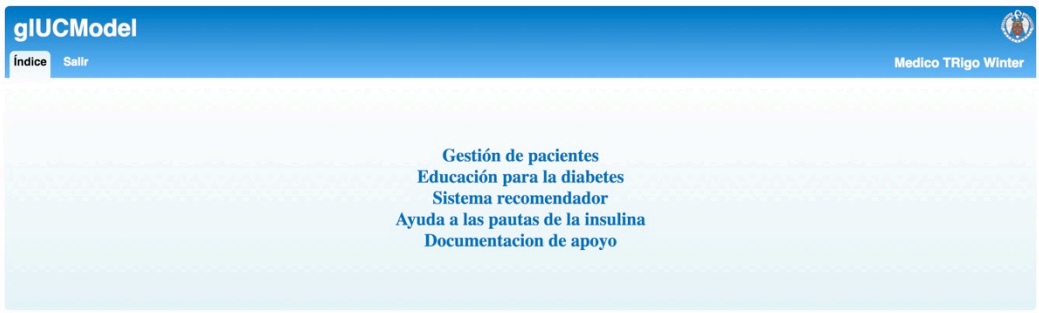

Figure 5: Old doctor index

There are also five options:

1. **Management of patients.** In an old version, this section looks like this [*Figure 6*]:

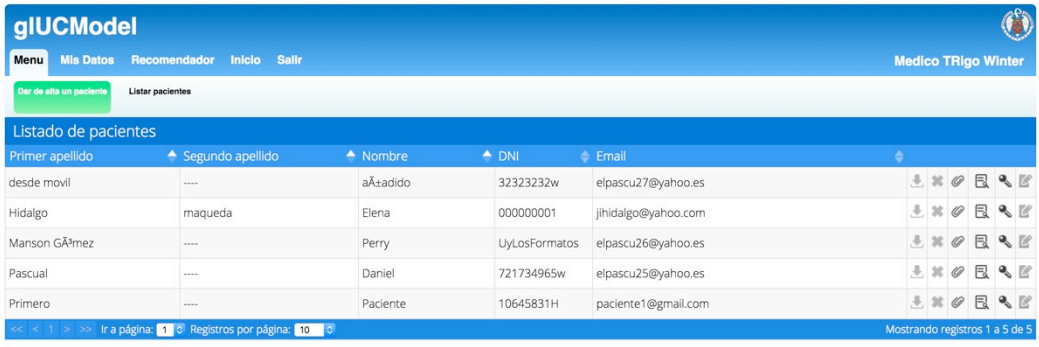

Figure 6: Old management of patient window

There is a separated tab to create add patient- in a new version this action can be realised by pressing a button on the top of a table. In addition, it is possible to download/delete/add a written authorization of a patient to manipulate his data, to get his records (to gain access to tables with medical data) and modify them, to change a patient´s password or to change another personal information. In a new version, the list of options is different [*Figure 7*]:

Figure 7: New management of patient window

In this case, we don´t talk about authorization- it´s supposed, that a patient couldn´t be registered in a system without his permission. A doctor is still able to see data records of a patient (1st icon) and modify his personal information, including password (4th icon). A patient can also be deleted from a system (5th icon). Now a doctor can manage all the evaluation cases, that were stored in a database for a concrete patient (2nd icon), and perform a new evaluation (3rd icon) - details of these operations will be explained later.

2. **Diabetes education.** The same functionality as for a patient- so, it´s just a link to the Moodle system;

3. **Recommender.** The first version has three options: to list all cases, to define a new one or to create a message. All these functions are implemented in a new system- to access those it´s necessary to choose second option (“Casos registrados”) on a patient list. A message, that was sent to a patient, will be represented like a new recommendation and will appear in his “mailbox”

4. **Advices about insulin.** The same functionality as for patients.

5. **Modify data.** A doctor may change his personal data- of course, it´s also possible to do this in a new version.

Database structure

The original version of “glUCModel” was working with a database that contained 20 tables. During the implementation of a new system, the following changes were made:

- **“Casos_formacion”**: this table was used to save additional parameters about a patient: for example, if he is a sportsman, if he is registered in moodle and/or insulin recommender system, if he drinks alcohol, etc. To simplify a new database, this table was eliminated;

- **“Casos_seguimiento”**: now every registered case is associated with a concrete patient;

- **“Destino_doc”** and **“Documentaciones”**: these tables were used to save a root to a saved document and all the relevant parameters. As the correspondent section was deleted from both menus, these tables were eliminated;

- **“Mensajes_usuario”**: all the messages that were sent to a patient by a doctor were sent here. In a new database, there is a table called **“Recomendacion_env”**, that is used for the same purposes;

- **“Momentos”**: this table was establishing a correspondence between a commonly used description (e.g. “after breakfast”) and concrete hours- deleted because of being redundant;

- **“Nivel_eje”** and **“Tipo_eje”**: these tables could be used in case of constructing charts in an application- but they were also redundant in a new version;

- **“Pruebas”**: the internal structure of this table was simplified- now it´s necessary to specify only a type of a medical test and the result;

- **“Tipo_ins”** and **“Tipo_pru”**: these tables were giving additional information about every kind of insulin and medical tests respectively. Eliminated to facilitate a database scheme;

- A table **“Usuarios”** was used like a base for a table **“Pacientes”**- however, some attributes were deleted because of changes in a functionality of an application;

- A new table called **“Doctores”** was created- it´s used to store personal information and access data about a doctor. The ID of doctor references to a “idDoctor” attribute of a previous table- that’s why the list on a doctor´s start page contains just his patients;

- A new table called **“Drugs”** was created: it is used to store a list of medicines that have been searched by doctors or patients

Other tables, that were not mentioned, were reused in a new database- and will be discussed in Chapter 5.

Conclusions

After performing an analysis of the original “glUCModel” system, it was decided to extend the functionality of this application in a new version. To do that, the next vectors of development were defined:

1. (**Main**) Implementation of recommender;
2. (**Main**) Implementation of export and import functionalities
3. (*Additional*) Redesign of the application to make it more user-friendly
4. (*Additional*) Change of database´s structure to reduce its volume

An implementation of these functions will be discussed in Chapter 5.

Chapter 3: Technologies and State of the Art

In the Chapter 2, I described those changes, that were made during the development of a new version of "glUCModel" [3]. Nevertheless, which technologies were used for an implementation? What do they serve for? Finally, what is a "recommender" and how can it learn something or propose solutions? The answers are given here.

Technologies

All the technologies that were used for the development of "glUCModel 2.0" can be separated into three blocks: Web design (to create an intuitive and user-friendly application), Object-Oriented Programming (mainly- for the development of recommender) and Database Design (management of data)

HTML

Hypertext Mark-up Language (or HTML [8]) is the standard mark-up language used to create web pages. It´s one of three core technologies that are used in a web development for creating pages or user interfaces (for both standard computers and mobile devices).

In fact, HTML is a set of special elements called "tags"- they can define parts of a web page (a heading, a paragraph, a footer, etc.), type of content (images, tables, blocks, lists…), position and/or style of different elements on a page (although it can be done with CSS, as described below). An explorer will interpret all these tags, so that the user could see a web page.

CSS

Cascading Style Sheets (or CSS [9]) is a style sheet language used for describing the presentation of a document written in a mark-up language. It´s the second core technology for web sites- applied to an HTML (or any other XML document), CSS permits to use different colours, fonts, layouts and other visual effects to different elements of a page.

As I said above, the style of elements can be defined directly in a .html file- but in big applications it´s more comfortable to define style rule in a separated .css file and then apply them to all the elements of a specific type or specified by an attribute (<id> or <class>).

Bootstrap

Bootstrap [10] is a free and open-source front-end web framework for designing websites and web applications. This is a combination of HTML- and CSS-based design templates, that can be used for a fast creation of a responsive web pages [11]- the main goal is to provide optimal viewing and interaction experience across a wide range of devices. Sometimes it may also have JavaScript extensions.

The front end of "glUCModel 2.0" is developed with Bootstrap- it was very useful, because this framework saved a lot of time, that was used for another purpose.

JS

JavaScript (or JS [12]) is a high-level, dynamic, untyped and interpreted programming language. This is the third core web technology, used in modern browsers as a script language to give interactivity to web pages.

JavaScript is used in AJAX (a set of web development techniques using different web technologies on a client-side to create asynchronous Web applications [13]). Once the data is modified, a page with AJAX doesn´t have to be completely reloaded to show the changes- it increases the performance of an application.

Apache Tomcat

Apache Tomcat (or just Tomcat [14]) is an open-source web server, which implements several Java EE specifications and provides a "pure Java" HTTP web server environment in which Java code can run.

As a huge part of an application is written in Java, I needed to "simulate" a server to check functionality of "glUCModel 2.0"

JColibri

JColibri [15] is a framework for developing CBR applications in Java, developed by GAIA group of UCM. It has different features: a persistence layer, retrieval methods, reuse and revision methods, maintenance components, evaluation methods, etc. This framework may be installed like a plugin for

Eclipse or it´s possible just to include jcolibri.jar file and dependent libraries into your project. There is also a jColibri Studio- a constructor of CBR apps.

Java

Java is a concurrent, class-based, object-oriented programming language, specifically designed to have as few implementation dependencies as possible [16]. This language tries to follow a WORA rule ("Write Once, Run Anywhere")- it means that a compiled Java code (typically compiled to bytecode) can run on any Java Virtual Machine.

MySQL

MySQL [17] is an open-source relational database management system. It´s a very popular choice of database for web applications and it is used in a LAMP (Linux – Apache – MySQL – PHP/Perl/Python) web application software stack. It´s possible to use MySQL via command line or via MySQL Workbench application.

Hibernate

Hibernate ORM (or Hibernate [18]) is an object-relational mapping framework for Java. A primary feature of Hibernate is mapping from Java classes to database tables and mapping from Java data types to SQL. To do this, it´s possible to use XML files [19] or Java Annotations [20].

Case-based reasoning: history, basic ideas and ways of use

Introduction

Every day we should make dozens of decisions- and usually when we choose between several solutions, we ask ourselves: "Ok, I have already faced a similar problem- how did I solve it?" This question is an essence of case-based reasoning. Janet L. Kolodner, one of the first researchers in this field, says that "CBR means using old experiences to understand and solve new problems" [21]. The main goal is to determine, whether some of past situations (called *cases*) are similar enough to a new one to be considered while solving a new problem.

There are two main kinds of CBR: *interpretative* and *problem solving*. In the first case, we will use previous situations (*precedents*) to create a justification and for the interpretation of a new case (that´s what usually a lawyer does). Another way is to adapt a previous solution to meet the requirements of a new case- that´s a typical task of an engineer [22]. In this project, I will use the second strategy: the decision will be based on a comparison between solved cases and a new one.

First CBR systems were created in the Yale University in 1980s under supervision of Roger Carl Schank, an American artificial intelligence theorist and a leading pioneer of AI and psychology in 1970s-80s. His model of dynamic memory [23] was the basis for Janet Kolodner´s system called CYRUS [24]. Computerized Yale Retrieval and Updating System stores and retrieves episodes in the lives of Secretaries of State Cyrus Vance and Edmund Muskie. When new events are added to its memory, CYRUS integrates them into memory along with the events it already knows about. CYRUS can then answer questions posed to it in English about the events it stores. Another example is Michael Lebowitz's IPP CBR system [25].

The international interest to CBR systems grew up in 1990s when an International Conference on CBR and workshops in different countries were established. Right now, case-based reasoning is widely used for constructing help-desk applications (those who provide information about products and services to the end users or customers) and health services [26].

CBR Cycle

Case-based reasoning is a continuous process, which can be divided (on a high level), in four steps [27]:

1. **Retrieve** the most similar case(s);
2. **Reuse** an information and/or knowledge to solve a problem;
3. **Revise** a proposed solution;
4. **Retain** the parts of this experience likely to be useful for future problem solving

First, we should define a CBR query, using the description of a current problem. This new case will be used to **retrieve** similar case(s) from a collection of previous situations. Then a new case and a retrieved one will be combined in a stage of **reuse** and a first version of solved case will be produced. After that, the solution will be **revised** (tested for success) by being applied into the real-world environment or evaluated by a human. If necessary, a solved case will be repaired. Finally, useful experience will be **retained** for future reuse- and a case base will be updated by a new learnt case or by a modification of an existing one. All this process can be described by a following scheme [*Figure 8*]:

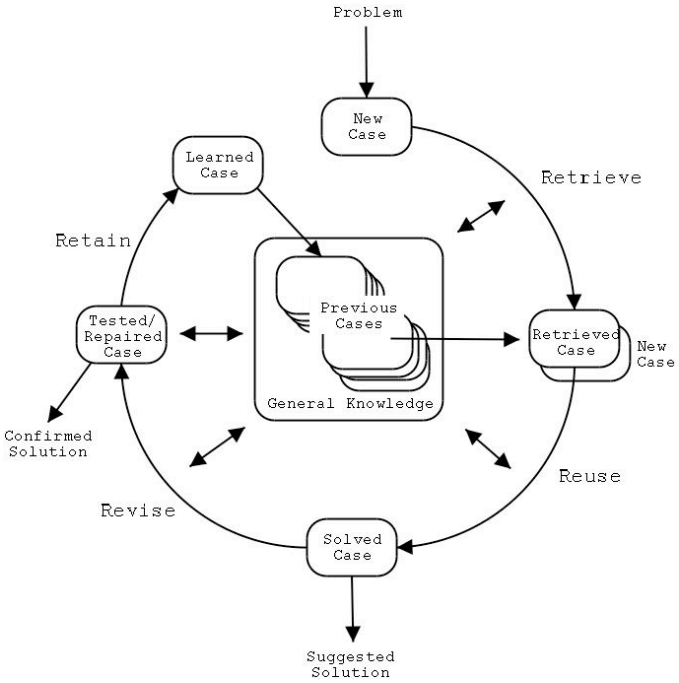

Figure 8: A diagram of CBR cycle [27]

Recommenders

Recommenders are one of the most typical examples of CBR applications. They try to learn user preferences over time and to automatically suggest solutions that fit the learnt user model. Recommenders may be classified on different parameters: number of end users, way of giving these recommendations, way of interaction with a user, etc.

First, there can be group or individual recommenders. In the first case, it´s necessary to create a unique recommendation for a group of persons, basing on and combining individual preferences of every one of them. It can be done in several ways: shuffling individual recommendations, considering personal evaluations of solutions or even creating a model of group and giving an individual recommendation for this model [28]. The other type is an individual recommender- they are working with a unique end user.

Let´s focus on individual recommenders. There are 2 different ways to suggest something to a person: basing on his profile/personal preferences or basing on evaluations of other users. In the first case, we are talking about recommender systems based on content, in the second one- about systems based on collaborative filtering [29].

Recommenders, which are based on content, use a base of cases and a special profile that determine an importance of attributes of a case. Usually a user may modify this profile and decide (1) which attributes are more important

from his point of view, (2) what kind of similarity functions he wants to apply and (3) how many cases should be retrieved.

The last important characteristic to be mentioned here is the way of interaction with a user [28]. There are two possibilities:

- **Single-shot:** the user receives a set of retrieved solutions- and he may choose or discard any of them;
- **Conversational:** a system interacts much more with a user to offer him the best solution. The main idea is that the recommender proposes a solution to user and waits for his opinion. Per the user´s decision, a system **modifies** a query, executes it and shows new solutions.

A user is able to modify the "weight" of parameters, if any of them are considered more important than others, and to select a number of retrieved cases. Two types of similarity functions are used:

- **Equal ():** a function that compares values of one attribute in a new case and in a stored one and returns 1 if they are equal or 0- if not;
- **Interval ():** a function that determines, if a value of an attribute in a new case is close enough to the value of this attribute in a stored case. The result can be calculated with the next formula:

$$Sim(x, y) = 1 - (\frac{|x\text{-}y|}{interval})$$

where:

x and **y** are values of an attribute in a new and an old case respectively

interval is the difference between maximum and minimum values that are allowed for this attribute

Both types of similarity functions are used in "glUCModel 2.0" ´s recommender. For example, "physical activity check" and "eye´s bottom check" parameters can have just two values (0 or 1), which means that these attributes will be treated with *equal()* function. At the same time, *interval()* similarity function is applied to IMC, level of HBA-1C [30] or high/low glucose levels due to variety of possible values.

Conclusions

Appeared in early 1980s, case-based reasoning became a very important concept, that is widely used in different science areas. One of the most typical examples of CBR applications are recommenders, that try to learn user preferences and give advices based on learnt information. Recommenders can vary on number of end users, way of interaction and giving advices, etc.

The one, which was implemented for “glUCModel 2.0”, is a single-shot individual recommender that is based on content. The user is able to modify the weight of parameters (by default, the weight is equal to 1, but it can be changed to 1.5), but can´t modify neither the quantity of retrieved cases (by default, it´s equal to 3), nor the case description attributes (these are automatically retrieved according to rules that are explained in Chapter 5).

Chapter 4: Design of “glUCModel 2.0”

This Chapter will be dedicated to the satisfactory explanations of the requisites of “glUCModel 2.0”. All the permitted conducts of future users will also be covered here to prevent problems during next phases. In this way, it will be possible to create an exhaustive description of application´s functionality.

It´s very important to identify all the models of future users, to find out their necessities and roles they will play in an application. This process will allow to decide, whether every one of functions should be accessible by every type of users, to delegate administrative rights and to improve the security of system.

Types of users

As in the previous version of “glUCModel”, there will be just two main types of users. The first one is Doctor- a physician that will evaluate patients´ situations and give recommendations. The other one is Patient- a person who suffers from diabetes and who wants to receive medical help by the application.

Now let´s see the functions of every type of users:

- **Doctor**

    1. **Register a new patient:** the doctor will be able to register a new patient- to do this, he/she will need to fill a questionnaire and submit it to the database;

    2. **Manage patients´ data:** the doctor will have access to the list of his patients and will be able to manage patients in general and any kind of medical data of a specified patient;

    3. **Send a message to a patient:** the doctor will be able to send a personalized message to a patient, indicating the reason (education, medical evaluation, etc.);

4. **Modify his/her own data:** the patient will have a possibility to change his personal data and submit it to database**;**

5. **Access to Moodle:** the doctor will have access to an e-learning platform called Moodle- he/she will be able to create/modify/delete themes and/or courses;

6. **Access to medical reference book:** the doctor will have access to a special guidebook, which contains information about different insulin medicaments, and will be able to register new medicaments;

7. **Work with recommender:** the doctor will have access to recommender- he/she will be able to create cases manually or evaluate a patient using the recommender. This option was not implemented in previous versions;

8. **Self-registration:** if a doctor is still not registered in the "glUCModel 2.0", he/she may fill a questionnaire and submit it to database;

9. **Password restore:** if a doctor will forget his password, he/she will be able to ask for a new one;

10. **Download/upload:** there is a possibility to upload files in .txt with data about weight or blood sugar level (these files must have a fixed format, which will be described in advance) or download a summary of any type of data. This option was not implemented in previous versions

- **Patient**

1. **Manage medical events in a database**: the patient will be able to add a new weight measurement, new glycemic or insulin value, register physical activity or upload data about latest medical reviews;

2. **Access to Moodle:** the patient will have access to an e-learning platform called Moodle;

3. **Access to medical reference book:** the patient will have access to a special guidebook, which contains information about different insulin medicaments;

4. **Work with recommender:** the patient will have access to some kind of mailbox so that he could read/delete given recommendations or ask for an evaluation;

5. **Download/upload:** the same functionality as for the doctor

Requisites

Technical restrictions

This web application is supposed to be used by doctors and patients, who will not pay too much attention to graphics and who will work with textual and numerical information. Furthermore, the speed and type of an internet connection may vary a lot. They can also use different browsers and devices. That´s why, this application has an optimized graphic part and uses up-to-date technologies to guarantee reliable communication and transfer of data between client and server.

It´s possible, that someone would like to use an application from the mobile phone- this aspect was also taken into account, so web pages will be automatically adapted to the screen size.

Use cases

Common use cases

Use case: login

- **Description:** user wants to login into the application and to gain access to his personal start page.

- **Actions to do:** on the main page, a user should introduce his email and the password in a special form. If there is a patient or a doctor with this email/password, a start page of a respective type of user will open. If there is no user with these credentials, a page will reload.

Use case: logout

- **Description:** user has ended to work with system and wants to logout

- **Actions to do:** to logout, a user should press on a word “glUCModel”, that is situated in a top left corner

Use case: manage medical data

- **Description:** there are five types of medical data, which may be used in the application: insulin injections, weight, glycemias, medical tests and exercises. It´s possible to make a new record, modify an existing one or

delete it for any of these types. The functionality is similar for all of them:

- **Actions to do:** if the user is a patient, he should just choose a type of medical data from a blue block at the foot of his main page. If he is a doctor, he should previously choose a patient, pushing “Ver detalles” button in an appropriate row. The user will see a table with records of selected type and will have different options. To create new record manually, he should push a “Crear nuevo registro” button- a questionnaire to fill will appear. To modify a record, it´s necessary to push an “Editar registro” button (looks like a sheet and a pencil). Finally, to delete a record the “Eliminar” button (a trash can) should be pushed. After that, user should push “Guardar” (or “Borrar” in case of deleting)- and the action will be performed. To add data automatically, user should *import data*- this use case is described below. He will also have an option to cancel an operation;

Use case: export data

- **Description:** user wants to save all the records of a concrete medical data type

- **Actions to do:** first, it´s necessary to gain access to a main page of a concrete patient: if the user is a patient, this will be his first page after login. In another case, he should previously choose a patient from his main page. To download, the user should select a type of medical data from a blue block at the foot of the page. On the next screen, it´s necessary to introduce the first and the last dates and press “Descargar”- a system will generate a file with all the data of a selected type of a patient

Use case: import data

- **Description:** user wants to upload records of blood sugar level, weight or physical activity

- **Actions to do:** first, it´s necessary to gain access to a main page of a concrete patient: if the user is a patient, this will be his first page after login. In another case, he should previously choose a patient from his main page. To download, the user should select a type of medical data (“Pesos”, “Glucemias” or “Ejercicios”) from a blue block at the foot of the page. Then he should push the button “Seleccionar archivo” and choose a file to upload. It should be 1 record per line and every record should have the next structure:

    - <yyyy-mm-dd hh:mm:ss> | <weight> for “Pesos”;

- <yyyy-mm-dd hh:mm:ss> | <value> | <moment> for “Glucemias”.
  NB: <moment> must have one of these values: “Antes del desayuno”, “Después del desayuno”, “Antes de comer”, “Después de comer”, “Antes de cenar”, “Después de cenar”, “Otros”;
- <yyyy-mm-dd hh:mm:ss> | <value> | <moment> | <description> for “Ejercicios”.
  NB: <description> must have one of these values: “Zancadas”, “Cinta”, “Dominadas”, “Remo”;

Use case: access to Moodle

- **Description:** user wants to access to Moodle (an e-learning system, that will be used for education of patients)

- **Actions to do:** from any page (except the index and pages dedicated to changes of password and/or personal data), user should push on the word “Moodle” on the top of a page. The system will redirect the user to a Moodle system

Use case: access to the medical reference book

- **Description:** user wants to add new drug to the database to have access to its data sheet or leaflet.

- **Actions to do:** the page with drug´s database can be accessed from different pages: doctor´s start page, patient´s start page or from any medical data page. In any case, it´s necessary to push the “Pautas de insulina” button on a top right corner. Drugs are represented in a table and the user can see two fields (name of product and marketing authorization owner) and three buttons: “download” icon serves to download data sheet, “list” icon is to download leaflet and “trash” icon is to delete a record from database. To add a new drug to the database, a user should enter it´s title- and a system will automatically propose different options. User can choose to add one of them, pushing the button “Elegir” or abort an operation, pushing “Cancelar”. In any case, the page will be reloaded

Use case: reset password (this functionality is NOT implemented)

- **Description:** user forgot his password and wants to restore it

- **Actions to do:** from the index page, user should push the “Contraseña perdida” button on a top right corner- a special form will be opened. User should write an email used for this system. If an email exists on a database, a new one-time password will be automatically generated and

sent to user. A user will also receive a recommendation to create new password instead of a generated one. At the same time, this password will substitute the old one in a database. If this email is not registered in a database, a message “Email not registered in a database” will be shown and user will be redirected to index page. If this email is registered, a message “New password was sent to you” will be shown and user will be redirected to index page.

Use cases only for patient

Use case: manage received recommendations

- **Description:** user wants to check all the recommendations he was given or messages he received, read or delete anyone of them

- **Actions to do:** click on “Recomendador” in the top right of the screen. If user wants to delete a recommendation/message, press “Eliminar” window. A dialog window will appear, asking to confirm an action. Pressing “Eliminar”, the user will delete this recommendation/message. “Cancelar” button will just close a dialog window.

Use case: ask for evaluation

- **Description:** user wants to ask his doctor to perform an evaluation of a current situation

- **Actions to do:** click on “Recomendador” in the top right of the screen. Under the message box, there is a button “Solicitar evaluación”. If an operation was performed correctly, a user will see a confirmation pop-up window

Use cases only for doctor

Use case: modify data

- **Description:** user wants to modify his personal data

- **Actions to do:** from any page (except the index), user should push on the phrase “Cambiar datos” on the top line. The system will redirect him to another page, where it will be necessary to fill in a questionnaire. Then the user should push the button “Confirmar nuevos datos”- the system will show to user a confirmation message and user will be redirected to his start page

Use case: manage patients

- **Description:** user wants to manage patients that are already included in a database or register a new patient

- **Actions to do:** to register a new patient user should choose a button “Crear nuevo registro”. A form will appear- user should fill it and press “Guardar”. It´s possible to cancel an action, pressing “Cancelar”. To edit information about a patient, user should press an “Editar registro” button- a new window will appear to give a possibility to change any information about a patient. Modifications will be saved with “Guardar” or will be discarded with “Cancelar” button. Pressing “Ver detalles” button, the user will see a profile of a selected patient (to make changes in medical records, please refer to a “manage medical data” case of use)

Use case: manage cases from recommender of a concrete patient

- **Description:** user wants to see all the cases of investigations

- **Actions to do:** from the list of patients, a doctor should push a sign with letter “i”- cases, that are registered for this patient, will be opened. In the first section of a page, user can find a table with all the cases of investigation. To see an associated recommendation, the user should press “***i***” button- a pop-up window will appear. To delete, user should press “Eliminar”. A new window will appear, asking to confirm an action- and user should press “Eliminar” (or “Cancelar” to abort an operation)

Use case: manually create new case of investigation

- **Description:** user wants to manually define a new case of investigation

- **Actions to do:** from the list of patients, a doctor should push a sign with letter “i”- cases, that are registered for this patient, will be opened. Then a doctor should click on “Crear nuevo registro”, fill all the fields and give appropriate recommendations and click on “Guardar”. A new case will be stored in a database

Use case: create new case of investigation with the recommender

- **Description:** user wants to define a new case of investigation for a patient, using the recommender

- **Actions to do:** from the list of patients, a doctor should push a sign with gears icon. On the next screen, a patient´s situation sum-up will appear. Basing on this information, doctor can choose parameters that he considers to be more important and generate a recommendation. Afterwards, a system will return 3 most similar cases and will propose choosing one of them or writing a personalized recommendation. Clicking on “Confirmar la selección”, a new user case will be stored

Use case: send a message

- **Description:** user wants to create a new message to patient

- **Actions to do:** push on “Casos registrados” of a patient, scroll the page until the end, fill all the fields and click on “Enviar mensaje al paciente”. The message will be sent and a patient will see it in his mailbox.

Conclusions

There were defined fifteen use cases for “glUCModel 2.0”. Three of them (evaluation with recommender, data upload and data download) are completely new and correspond to two main vectors of development that were defined in *Conclusions* of Chapter 2. The other twelve use cases describe actions that could be performed in initial version of the application- however, their flows suffered minor changes in order to comply with technical requisites.

Chapter 5: Implementation of “glUCModel 2.0”

Database structure

When all the requisites and functions are defined, it´s time to create a structure to store all the information we will need. Our database will be created with MySQL, as it is powerful enough and open-source. On the next page, there is a complete scheme of a database structure [*Figure 9*]:

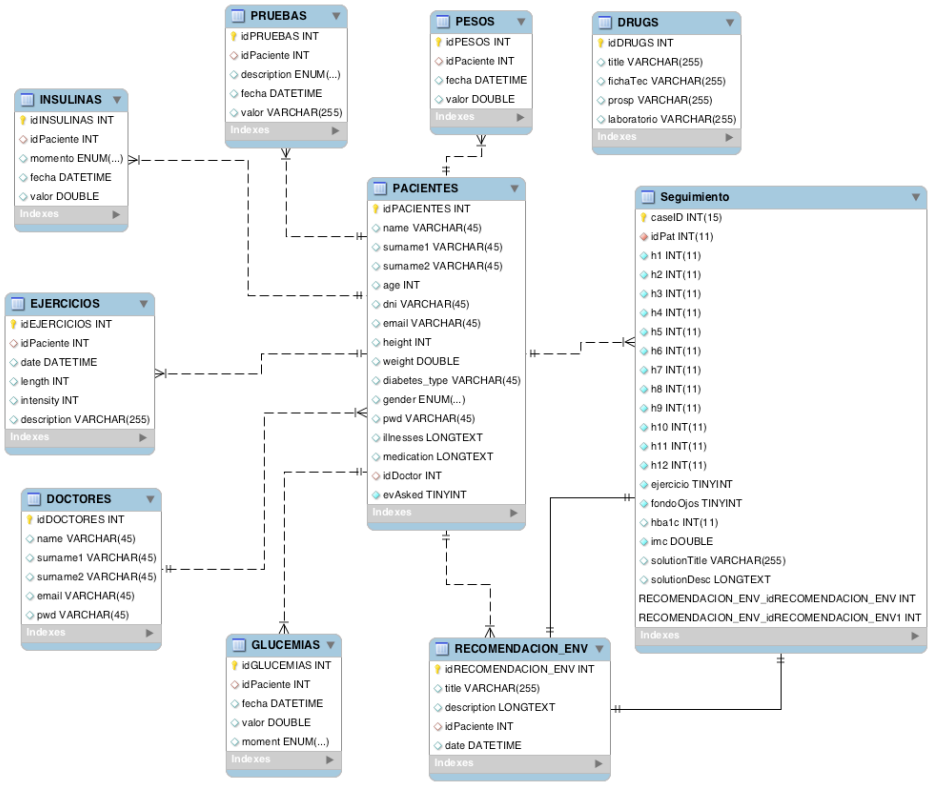

Figure 9: New database structure

Description of tables in the database

1. The first block of tables can be called “users”: it is composed by “PACIENTES” (patients) and “DOCTORES” (doctores) and stores all the personal data of them

   PACIENTES: a table that stores personal information of patients:

   - *idPACIENTES*: a primary key with auto-increment enabled;
   - *name*, *surname1* and *surname2* of patient;
   - *age*;
   - *dni*;
   - *email* (will be used for login);
   - *height*;
   - *weight*;
   - *diabetes_type* (will be defined by doctor);
   - *gender* (to choose between “Hombre” (male) or “Mujer” (female));
   - *pwd*: a password (will be used for login);
   - *illnesses*: description of illnesses of this patient;
   - *medication*: description of medicines, that this patient is taking;
   - *idDoctor*: identifier of his doctor (reference to idDOCTORES of table “DOCTORES”)
   - *evAsked*: a trigger that shows, if a patient asked for evaluation

   DOCTORES: a table that stores personal info of a doctor:

   - *idDOCTORES*: a primary key with auto-increment enabled;
   - *name*, *surname1* and *surname2* of doctor;
   - *email* (will be used for login);
   - *pwd*: a password (will be used for login)

2. The second block is called “medical data” and is composed by “PESOS” (weights), “PRUEBAS” (medical tests), “DRUGS” (trademarks of medicines), “GLUCEMIAS” (glycaemias), “INSULINAS” (insulin injections) and “EJERCICIOS” (exercises)

   PESOS: a table that stores all the weight measurements:

   - *idPESOS*: a primary key with auto-increment enabled;
   - *idPaciente*: identifies a patient that stored his result (reference to idPACIENTES of table “PACIENTES”);
   - *fecha* (date of measurment);
   - *valor*: the result of measurement

PRUEBAS: a table that stores all the medical tests:

- *idPRUEBAS*: a primary key with auto-increment enabled;
- *idPaciente*: identifies a patient that stored his result (reference to idPACIENTES of table “PACIENTES”);
- *description*: choose a type of test;
- *fecha* (date of measurment);
- *valor*: the result of test;

DRUGS: a table that stores all the trademarks of insulin:

- *idDRUGS*: a primary key with auto-increment enabled;
- *title*: trademark of a medicine;
- *fichaTec:* link to download data sheet
- *prosp:* link to download a prospect
- *laboratorio:* name of a company that produces this medicine

GLUCEMIAS: a table that stores all the glycemia measurements:

- *idPESOS*: a primary key with auto-increment enabled;
- *idPaciente*: identifies a patient that stored his result (reference to idPACIENTES of table “PACIENTES”);
- *fecha* (date of measurment);
- *valor*: the result of measurement;
- *moment*: user should choose the moment of a day

INSULINAS: a table that stores a history of insulin injections:

- *idINSULINAS*: a primary key with auto-increment enabled;
- *idPaciente*: identifies a patient that stored his result (reference to idPACIENTES of table “PACIENTES”);
- *fecha* (date of injection);
- *valor*: quantity of insulin injected;

EJERCICIOS: a table that stores all the physical activity:

- *idEJERCICIOS*: a primary key with auto-increment enabled;
- *idPaciente*: identifies a patient that stored his result (reference to idPACIENTES of table “PACIENTES”);
- *date* (of exercise);
- *length* (in minutes);
- *intensity*: from 1 to 10;
- *description*: a brief explanation of exercise;

3. The third block is “recommender tables” and is composed by “CASOS_SEGUIMIENTO” (cases of investigation), “PETICIONES” (petitions), “RECOMENDACION_ENV” (recommendations sent to a patient) and “RECOMENDACIONES” (recommendations)

   CASOS_SEGUIMIENTO: a table that stores all the investigation cases:

   - *caseID*: a primary key with auto-increment enabled;
   - *idPat*: a reference to a patient;
   - *h1*: hyperglycemia before breakfast (amount during a week);
   - *h2*: hypoglycemia before breakfast (amount during a week);
   - *h3*: hyperglycemia after breakfast (amount during a week);
   - *h4*: hypoglycemia after breakfast (amount during a week);
   - *h5*: hyperglycemia before lunch (amount during a week);
   - *h6*: hypoglycemia before lunch (amount during a week);
   - *h7*: hyperglycemia after lunch (amount during a week);
   - *h8*: hypoglycemia after lunch (amount during a week);
   - *h9*: hyperglycemia before dinner (amount during a week);
   - *h10*: hypoglycemia before dinner (amount during a week);
   - *h11*: hyperglycemia after dinner (amount during a week);
   - *h12*: hypoglycemia after dinner (amount during a week);
   - *ejercicio*: TRUE if a patient has any registered activity in EJERCICIOS table during the last 2 weeks;
   - *fondoOjos*: TRUE if a patient has any registered eyes check in PRUEBAS table during the last year;
   - *hba1c*: the last result (if there is any) of HBA1C test from PRUEBAS table;
   - *imc*: a coefficient of mass equal to “last weight/height in metres * height in metres)”;
   - *solutionTitle*: title of a recommendation that was given;
   - *solutionDesc*: description of a recommendation that was given

   RECOMENDACIONES: a table that stores all the recommendations:

   - *idRECOMENDACION*: a primary key with auto-increment enabled;
   - *title* (of recommendation);
   - *description*: the text of recommendation;
   - *priority*: from 1 to 3;
   - *reason*: the main purpose of recommendation

   RECOMENDACION_ENV: a table that stores sent recommendations:

   - *idRECOMENDACION*_ENV: a primary key with auto-increment enabled;
   - *idPaciente*: identifies a patient that stored his result (reference to idPACIENTES of table “PACIENTES”);

- *title*: title of a given recommendation;
- *description*: description of a given recommendation;
- *datetime* (when it was sent);

Frontend of the application

When we talk about software design, we may separate our application in two big parts: front-end and back-and. The first one is related to all the components that are manipulated by the user- in case of "glUCModel 2.0" it includes web pages, written in with HTML and CSS, and JS-files. The back-end is a server-side code that processes data- in this project this code is written in Java and will be explained later.

This application is created for doctors and patients- these are typical computer users, that will need an easy and user-friendly interface. I used a framework called Bootstrap, which is great to create complex web sites without deep knowledge of web design. Although it´s supposed that "glUCModel 2.0" would be used on PCs or laptops, all the pages can be resized to the screen of tablet or smartphone.

Although all web pages are written in HyperText Markup Language, they are saved with .jsp extension instead of .html. It happens because the backend of this project is created with JSP technology [31]. In this way, a developer is able to create dynamic web pages.

Mockups

As I was working on a web application, it was important to create an attractive index page- it will be the first thing that the final user will see. It will be used for login and few more operations [*Figure 10*]:

Figure 10: index.jsp

On the right, there is a validation form- a user (doctor or patient) should introduce his email and password: if this pair exists in a database, he will be redirected to an appropriate start page.

If a new doctor wants to use an application, he should submit basic information by the next form [*Figure 11*]:

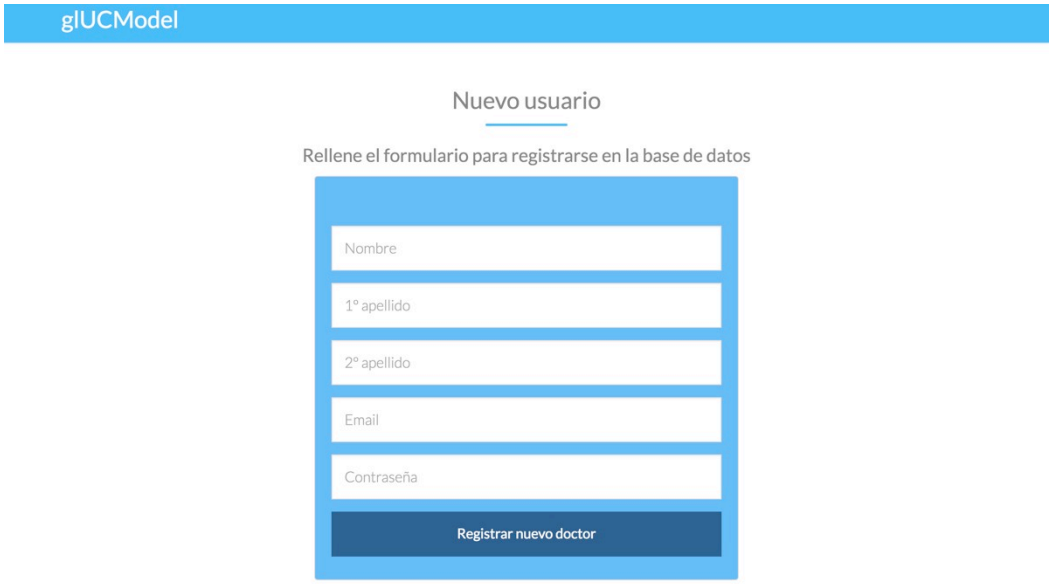

Figure 11: New user window

If the user lost his password and wants to restore it, there is an option of "Contraseña perdida" (*Figure 12*; this functionality is NOT implemented):

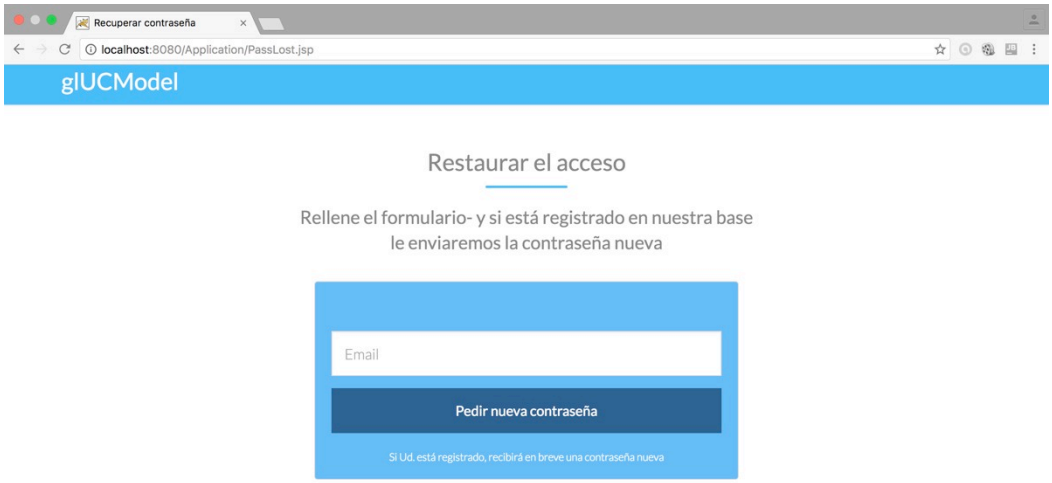

Figure 12: Password lost window

Doctor pages

If a doctor is logged in, he will see something like this [*Figure 13*]:

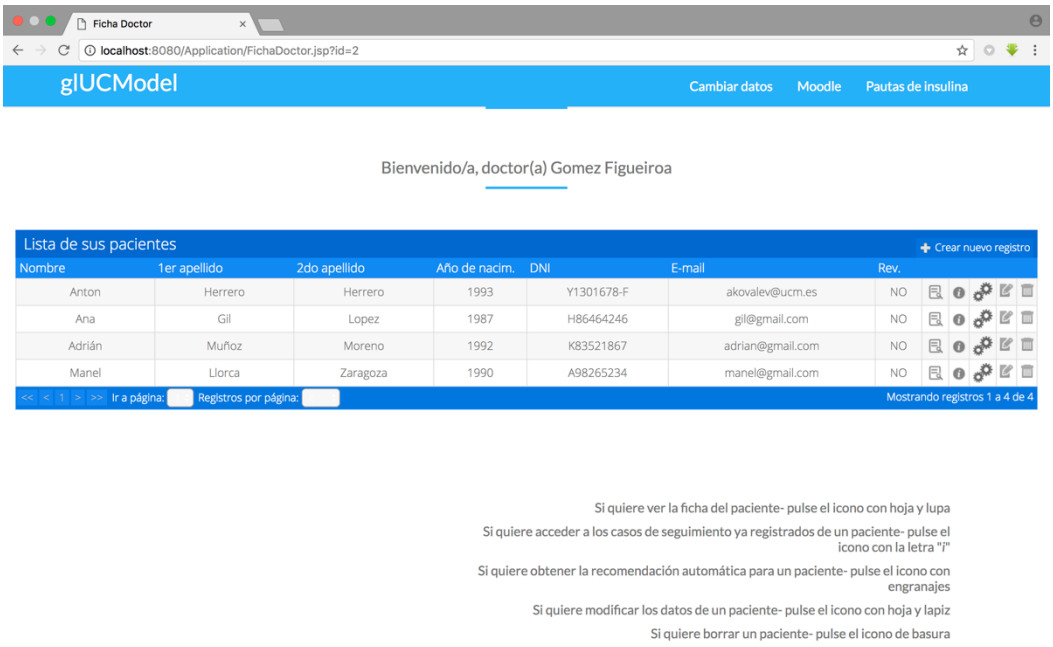

Figure 13: Doctor start page

Patients are shown in order of being registered (also, a patient cannot register himself- his doctor should fill in a form, which will appear on pressing "Crear nuevo registro"). For every one of them it´s possible to do five things (icons for left to right):

- "Ver ficha del paciente"- a doctor can see a page with information about a patient (an example will be provided in a next section);
- "Casos registrados"- show all cases of evaluation, that have already been registered for this patient [*Figure 14*]:

Figure 14: Registered cases of a patient

Here it´s possible to create a new case (filling all the fields manually), edit any of them or delete. Cases are ordered by date (the newest- first). “Ficha del doctor” link will return a doctor to his start page;

To see a title and text of a related recommendation, it´s necessary to press “***i***”- a pop-up window will be shown [*Figure 15*]:

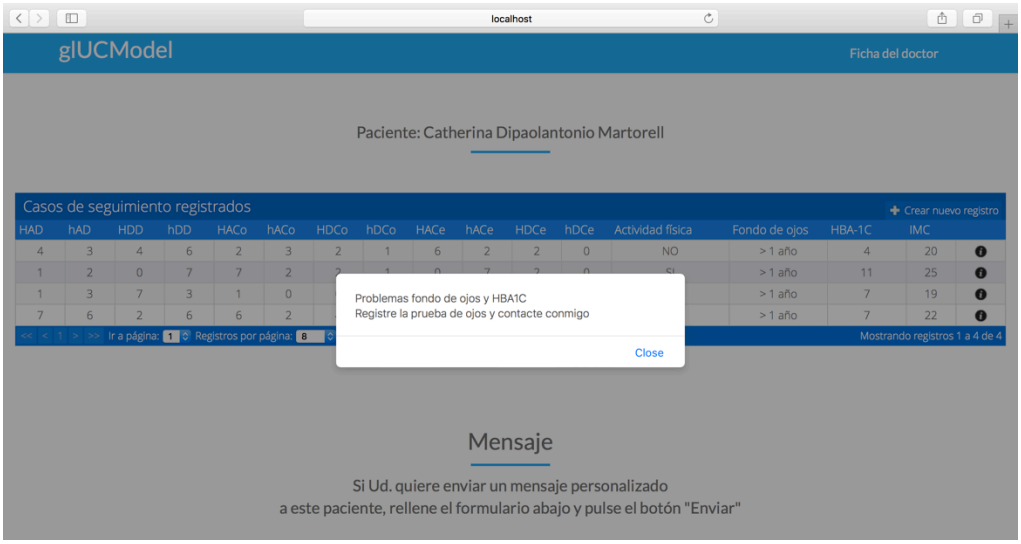

Figure 15: Pop-up window

Doctor also can send a personal message to a patient. To do this, it´s necessary to complete a form below registered cases´ table [*Figure 16*]:

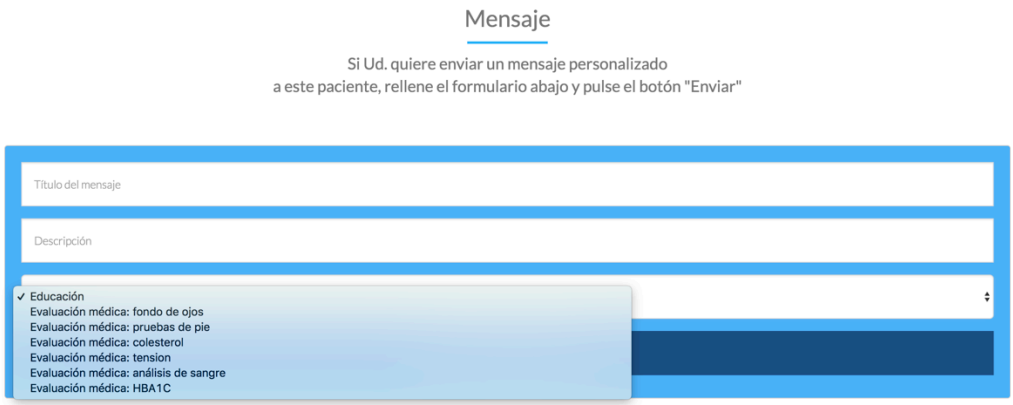

Figure 16: Personal message

Doctor may also change his personal data- for this, it´s necessary to press “Cambiar datos” [*Figure 17*]:

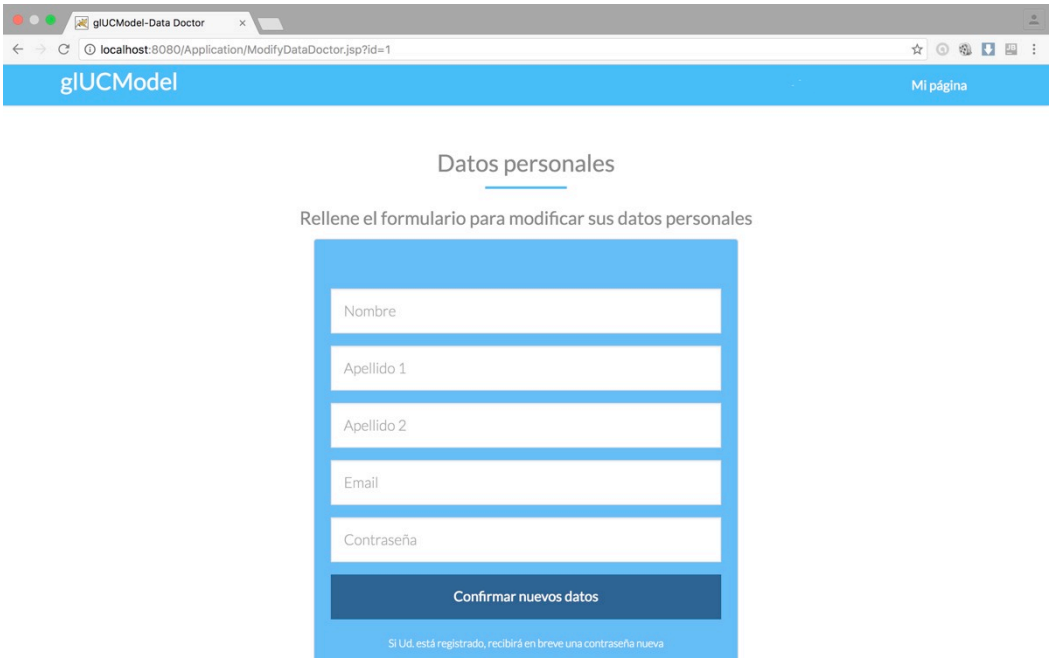

Figure 17: Change personal data

Patient pages

If a patient is logged in, he will see the next page [*Figure 18*]:

Figure 18: Patient start page

A wide zone on the left of a page shows basic information about a patient (name, DNI, diabetes type…). On the right, there is a block with charts, which represent dynamics of weight, blood sugar level (last 40 values for both diagrams) and IMC. Pointing at first two diagrams, a user can see a concrete

value (for example, 93 kg or 80 mg/dl). An IMC line has 4 zones: underweight (lower than 18.5, yellow zone), normal range (between 18.5 and 25, green zone), overweight (between 25 and 30, orange zone) and obesity (higher than 30, red zone). A wide orange line shows a current IMC of this patient and a thin vertical line is a perfect result (24). Finally, at the foot of a page there are links to five types of data, which are saved for every patient: insulin injections, weight, glycaemias, medical tests and exercises.

For example, if we want to work with exercises, the next page will open [*Figure 19*]. Any record can be modified or deleted. “Mi página” link will send you to a start page of a patient:

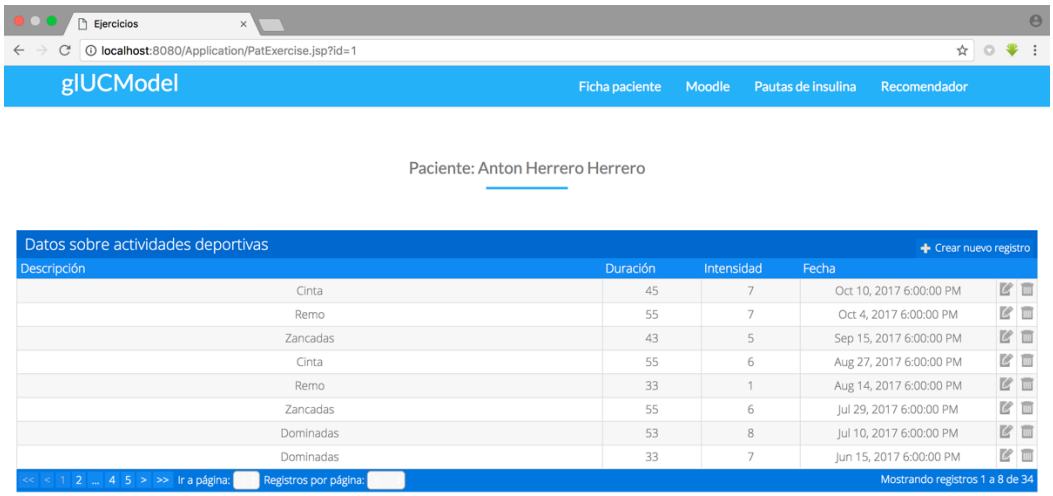

Figure 19: Exercises

As it was described in use cases description (Chapter 4), there are two ways of data import. The first one is manual and is implemented for any type of medical data. To do it, user should press “Crear Nuevo registro”- and the following form appears [*Figure 20*]:

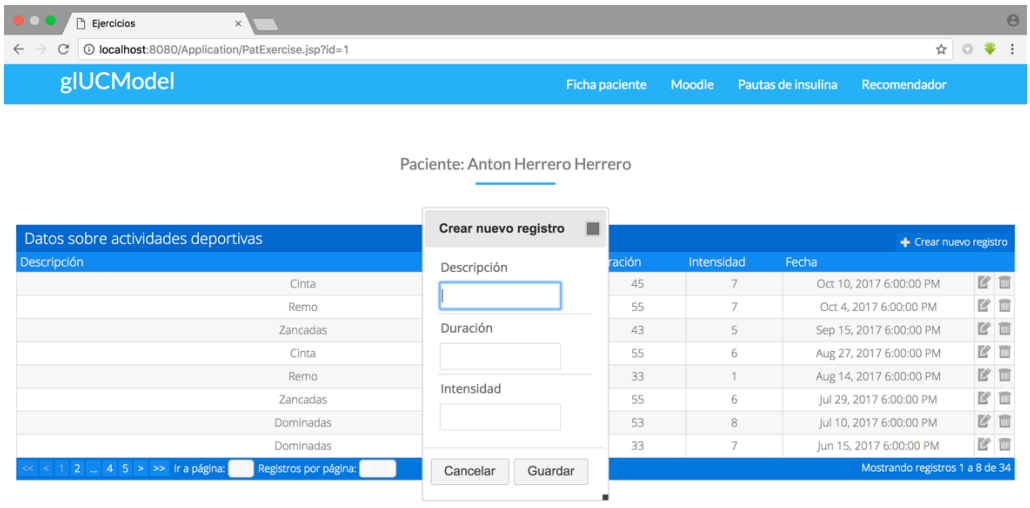

Figure 20: Register new activity

Below the table with records, there is a block for data export- user should introduce two dates and a system will generate a .txt file with records of selected type of this patient that were saved in a selected period of time [*Figure 21 - left*].

For 3 types of data ("Pesos", "Ejercicios" and "Glucemias") it is also possible to import records automatically. To do this, a user should click on "Choose file" button inside "*Importar datos*", select an archive and push "Importar" [*Figure 21 - right*]:

Figure 21: Export/import of data

A patient is responsible to upload new records (manually or automatically) to the system and he will not be notified by the application if the latest information was provided a long time ago. However, as it will be shown in advance, a recommender indicates the doctor if the latest registered eye´s bottom check (or latest physical activity) was performed several months ago- so, the doctor can mention this in recommendation´s description or just write a separate message to a patient.

Finally, if a patient presses a "Recomendador" button, he will see the next page [*Figure 22*]:

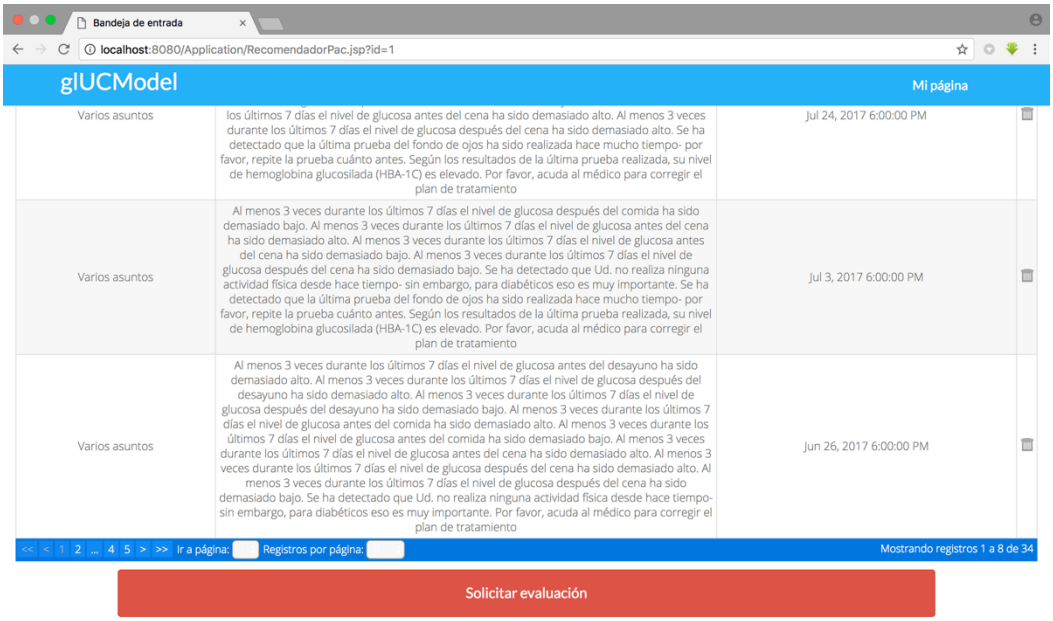

Figure 22: Mailbox of the patient

It looks like a typical mailbox: when a doctor registers a new case of study, he also specifies a title and description of an appropriate recommendation. As a case is registered for a concrete patient, there is everything to create a new "Sent recommendation"- and a patient will be able to see it here. A recommendation cannot be modified, just deleted. Messages that were sent by the doctor will also appear here. Under the table, there is a button "Solicitar evaluación"- if a patient presses it, an automatic request will be sent and a doctor can see it on his main page ([*Figure 13*], column "Rev.").

Common functionalities

Both patients and doctors have access to "Moodle" and "Pautas de insulina" options.

Moodle [32] is the learning management system, which is widely used to create online courses. In case of "glUCModel", a correspondent educational space was created by researchers of DACYA [2]- so the implemented system only contains links to access to it. User can do this in two ways:

- pressing "Moodle" button on his main page;
- pressing "Moodle" button on pages with medical data;

In any case, the next window appears:

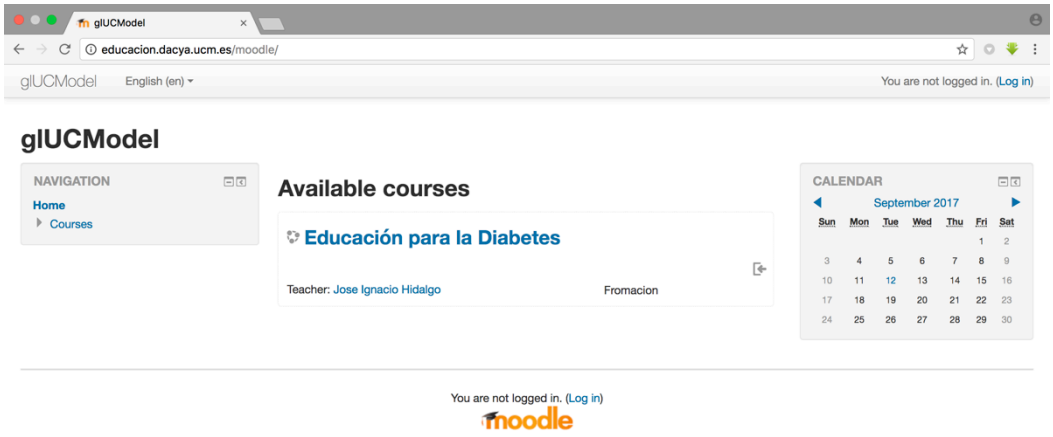

Figure 23: Moodle start page

An option "Pautas de insulina" lets user access the database of medicines. On a start page (which is accessible from both patient´s and doctor´s modes), there is a table with all the medicines that were saved- and a user has

access to data sheet (download icon) and leaflet (list icon) of a medicine. There is also a possibility to delete a medicine (trash icon) from database, as shown below [*Figure 24*]:

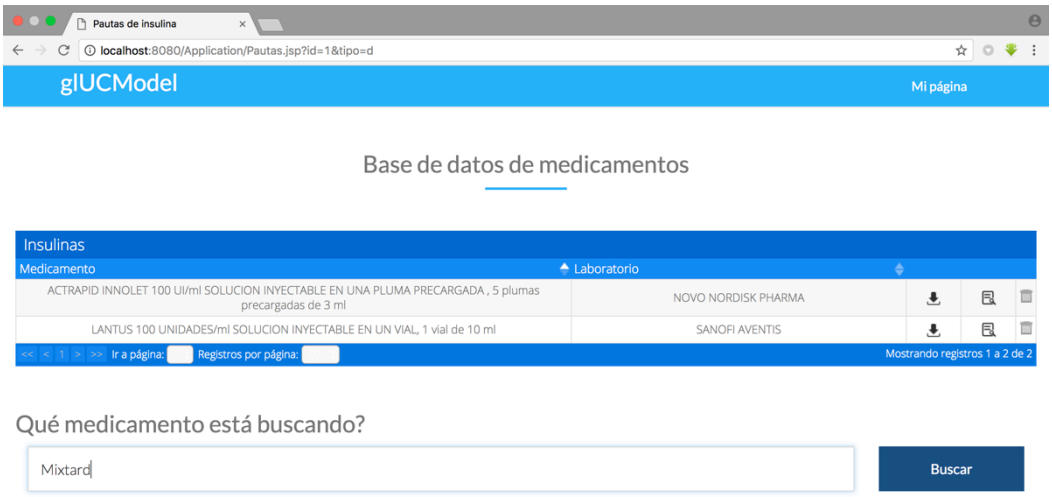

Figure 24: Medicine´s database

If the user wants to add new medicine to the base, he should introduce its title to the search field and push the button “Buscar”. An application connects with the database of the Spanish Agency of Medicinal Products and Medical Devices and shows the list of all medicines that contain a title that was introduced. For example, if a user needs “Mixtard”, he will see the next screen [*Figure 25*]:

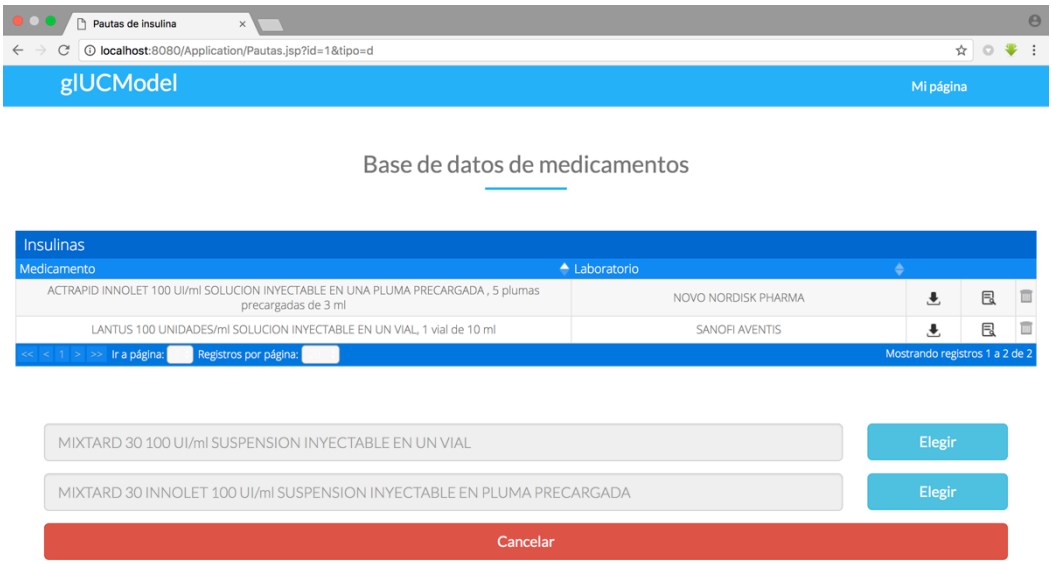

Figure 25: Search result of a medicine

The user can abort the process (pushing on “Cancelar”) or choose one of these variants. In this case, a page reloads and the medicine is added to the base [*Figure 26*]:

Figure 26: Search result of a medicine

Backend of the application

As it was mentioned, the front-end is composed of JSP pages. To run and deploy them, it´s also required to have a compatible web server with servlet container- I used Tomcat 7.0. Finally, a dozen of JavaBeans were created- these are typical Java classes that encapsulate several objects into a single one.

We can say that the backend of “glUCModel 2.0” follows the model-view-controller architectural pattern, where JavaBeans are storing a retrieved data, JavaServer Pages are responsible for the interaction with user and servlets (as a controller) are processing input information and sending commands to model. All this process can be described with a following scheme [*Figure 27*]:

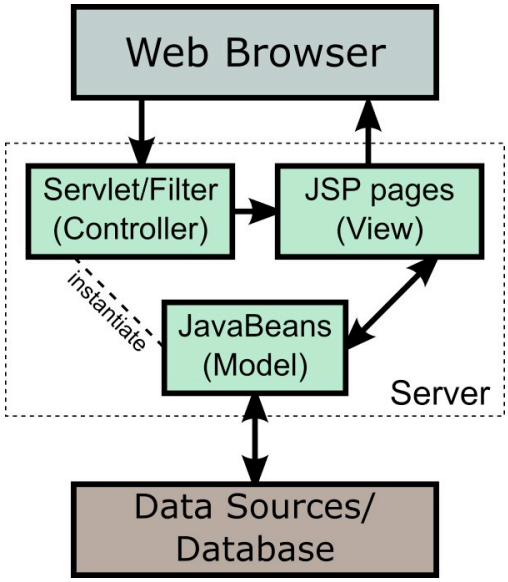

Figure 27: High-level scheme of application architecture

General structure of packages

All the Java classes are grouped (by their functionality) in 5 packages:

- **beans:** in this package there are all the JavaBeans and appropriate .hbm.xml files (in the next section, we will see what do they serve for);
- **crud:** basically, these are special classes, that perform operations with database. CRUD is an acronym of "Create-Read-Update-Delete"- four basic functions of persistent storage. There are also additional functions, that can be needed while inserting a new record;
- **database:** contains one Java class and one XML file, that are used for establishing a hibernate connection;
- **recom:** contains code of a recommender- main class, 2 JavaBeans and appropriate .hbm.xml files;
- **servlet:** this package has all servlets (classes that work like controllers)

Hibernate: configuration and implementation

As we already know, Hibernate is an object-relational mapping framework for Java: the main function is to declare relations between attributes in JavaBeans and SQL tables. If we want to use Hibernate in our application, the first archive to create is "hibernate.cfg.xml"- a configuration file (*Appendix A: Fragment 1*).

This file has two principle sections. First, we define properties for a connection to the database: driver, URL, user, password, dialect, etc. And then we should mention all the .hbm.xml files – they will be used for mapping between Java Classes and database tables. For example, there are four attributes in a Pesos table:

- id of the record;
- id of patient;
- date of measurement;
- value.

A JavaBean (*Appendix A: Fragment 2*) has setters and getters for all these attributes and a constructor, which will permit us to store a record from a database. While creating an appropriate .hbm.xml file (*Appendix A: Fragment 3*), we should declare a name of a respective JavaBean, table and name of database, where we will retrieve data from. Then there is a special tag **<id>**, used to specify an attribute, that was a primary key in the SQL table. For every attribute it´s necessary to specify its name in JavaBean, name in SQL and type in SQL. There can be also additional parameters, such as "precision" (because we have a DOUBLE value), "not-null", etc.

One of the most important concepts in Hibernate is "session"- the main runtime interface between a Java application and Hibernate. This is the central API class abstracting the notion of a persistence service [33]. An application has a special class called "HibernateUtil" for creating sessions when any CRUD class will need to perform an operation (*Appendix A: Fragment 4*).

For example, to perform a "delete" operation over a record of weight (*Appendix A: Fragment 5*), we had to do four steps:

1. Create a session and start a transaction;
2. Look for a record that has a "pesId" attribute equal to a passed parameter, and retrieve it.
3. Give an order to delete this record and to commit this transaction.
4. Flush (synchronize the persistent store with persistable state held in memory) a session and close it.

Java classes

CRUD package

All the CRUD classes have similar methods, that are used to:

- Receive a list of objects of determined type (for **R**ead operation);
- Receive a number of objects of selected type and the last used ID;
- **C**reate, **U**pdate and **D**elete registers

Some additional methods can also be included. *Appendix A: Fragment 6* is an example of this type of classes- in this case, all operations are performed with records of weight

Database package

This package contains two files- a HibernateUtil.java (we´ve talked about it above, while discussing a "session" concept) and a databaseconfig.xml (*Appendix A: Fragment 7),* that has five pairs of parameter and a relative root of a correspondent file:

- Hibernate configuration file;
- File with mapping of a case description class;
- A case description class;
- File with mapping of a case solution class;
- Case solution class

Servlets package

In this application, servlet classes take part of a controller. The core and only method here is a doPost that has two arguments: HTTPServlerRequest and HTTPServletResponse. Once a servlet is called, it receives all the necessary information in "request"- and everything that will happen depends on parameters of this request, so there will be many operations like "request.getParameter()"

A servlet should determine, which action was requested. There are four options (here I will talk about a servlet for patients (*Appendix A: Fragment 8*) - however, all the other Contr* classes work on the same way):

- **List:** it´s requested to show all the records of an appropriate type. To do this, he will send call an appropriate method in a correspondent CRUD class. It´s import to say, that we need just a list of patients of a concrete doctor- so, one of the parameters, that will be passed, is an ID of a doctor;
- **Delete:** it´s asked to delete a record from a system. A servlet receives an ID of an object, that should be deleted (in our case- an identifier of a patient). A CRUD method *deletePac* will be called and an ID will be passed as an attribute;
- **Insert** and **update** two very similar operations, when it´s necessary to create a new Object of a correspondent JavaBean class (a Patient) and to fill it with information, which is also passed as request parameters. After that:
    - if it´s a new patient: we should ask a CRUD class to calculate an ID and then- to save an object;
    - if we are updating an information: to ask a CRUD to modify an object

In any case, after an operation was performed, it´s necessary to give a response. As I used a jTable plugin [34] to create all these tables, it should have a special format and to be written in JSON. Finally, this response will be sent back to a JSP- and changes on a view will be performed.

Another important thing: as we are using Servlets, these should be declared in a special file web.xml (*Appendix A: Fragment 9*). For every servlet, there will be four fields:

- Servlet
  - Servlet-name: name of a class;
  - Servlet-class: relative root of a class;
- Servlet-mapping
  - Servlet-name: once more, a name of a class;
  - URL-pattern: this one will be used in a correspondent JS-file to specify, which servlet should treat a request

JSPs

One of the biggest advantages of using JSPs instead of HTML files is a possibility to add Java code inside the page. It can be done in different ways: using directives (see *Appendix A: Fragment 10*), delimiters, scriptlets or JSTL [31] (*Appendix A: Fragment 11*).

For example, directives and JSTL constructions were used to show a basic information about a patient on his main page [*Figure 28*]:

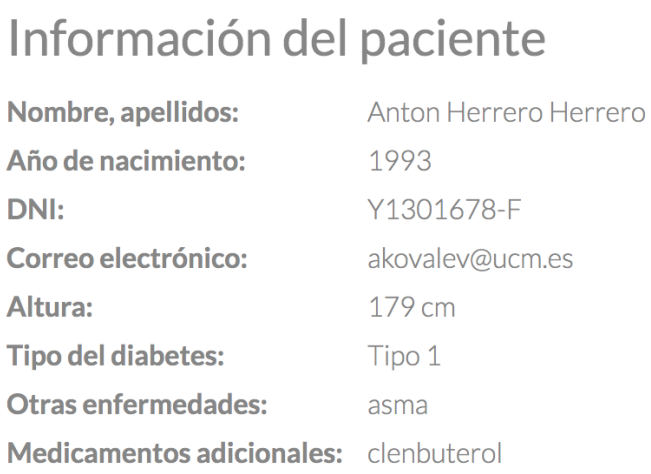

Figure 28: Patient information block

Recommender

The most important stage of this project is an implementation of a recommender. As it was discussed in Chapter 3, they can be classified in different ways- the recommender that was implemented is an *individual* one and is *based on content*, which means that it will work with one end user and the base of cases will be needed.

The first step to create a new CBR application is to **configure** it (*Appendix A: Fragment 12*):

- Initialize a Database connector (in this case, the configuration is written in “database/ databaseconfig.xml” archive);
- Create a LinealCaseBase to represent all the cases.

The next phase is **precycle()**- an operation, that will be performed just once and that is usually used to load the case base and, if necessary, make difficult and/or expensive computations (*Appendix A: Fragment 13*). For this we need to initialize a case base with a connector (that was defined earlier) and to call a getCases() method. Every case may have four components: description of a problem, solution of a problem, result of applying a solution and justification of the solution. Just the first one is obligatory- other can be left empty. In “glUCModel 2.0” the description and the solution are saved

The next step is to **obtain a CBRQuery**- a description part of a new case. In this version of application, arguments for a query are retrieved in an automatic way, following the next rules:

1. There are 6 different moments of time (before (**ADe**)/ after (**DDe**) breakfast, before (**ACo**)/ after (**DCo**) the lunch and before (**ACe**)/ after (**DCe**) the dinner) and 2 types of aberration (**Altas** means that the level of glucose was more than 100 mg/dL, **Bajas** means it was lower than 70 mg). An application retrieves all the measurements of glucose during last 7 days- and shows this information to a doctor;
2. A very similar procedure is performed to evaluate physical activity (a system checks that a patient has at least 1 register during the last month), existence of eye´s bottom (at least, once in 10 months) and the level of HBA-1C [30]

The description is represented as it´s shown below [*Figure 29*]:

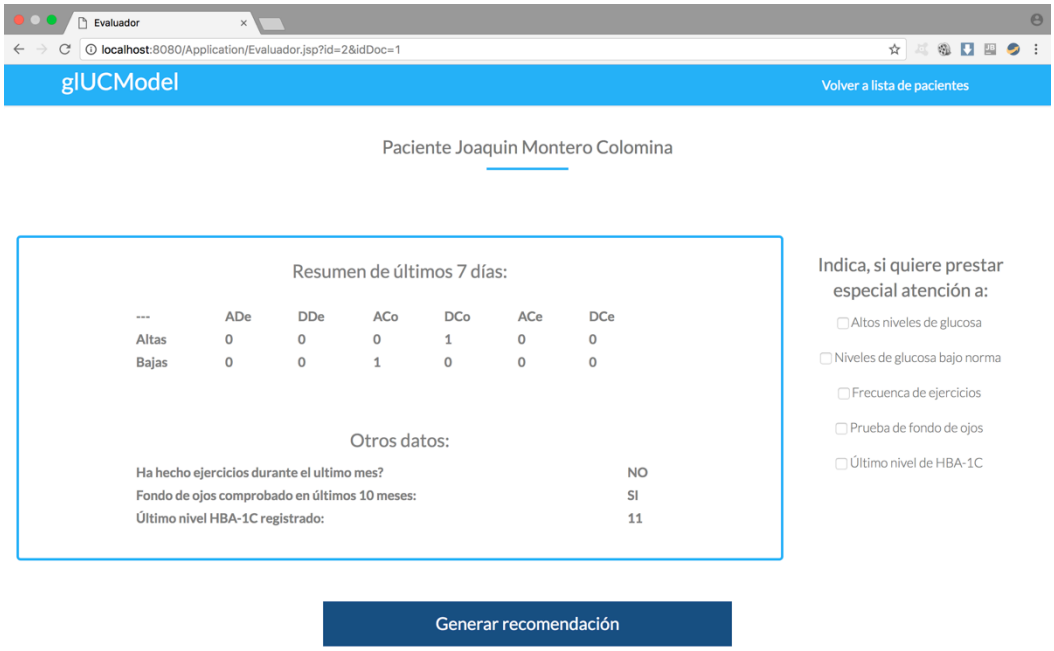

Figure 29: Situation sum-up

These parameters will be used in the next phase of a CBR cycle, which is called **cycle()** and which is one of the most important in the application (*Appendix A: Fragment 14*). The first part is dedicated to creation and personalization of an object called "config"- this is a filter, which would be applied to compare a new case and all the stored ones. It´s necessary to define two types of similarity functions, that *config* will use:

- *Global similarity function*: for example, the CaseDescription case component is a compound attribute that is composed by several simple attributes (like "h1", "imc", etc.) It was decided, that an *average* function will be used;
- *Simple similarity functions*: these are specified for every simple attribute- in these applications, it will be *equal* or *interval* functions (we´ve already talked about them in Chapter 3);

It´s also possible to define the *weight* of every attribute by checking or unchecking options in the right part of a screen [*Figure 25*]:

1. High glucose levels;
2. Low glucose levels;
3. Physical activity check;
4. Eye´s bottom check;
5. Level of HBA-1C

If an option is selected, a weight of attribute is changed to 1.5.

When a config file is ready, a special JColibri class called NNScoringMethod will perform one of the most important operations of a

cycle(): it will compare **stored cases** with a **query** (a new case) basing of **configuration** that we defined on a previous step (*Appendix A: Fragment 15*). The most similar K cases (coefficient K is three, although in many applications it can be also selected while defining a configuration) will be retrieved and shown.

In this moment, a doctor will see proposed solutions in the next window. As it is a *single-shot* recommender, the user will just be able to choose one of variants that were given or to write his own recommendation [*Figure 30*]:

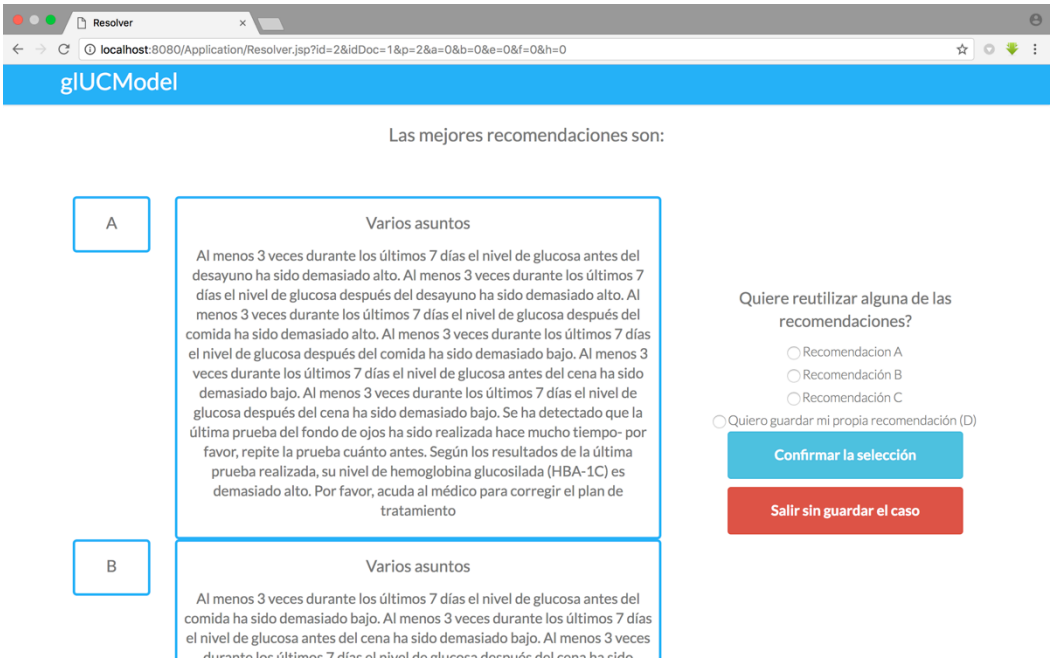

Figure 30: Window for choosing a solution

If the doctor presses “Salir sin guarder el caso”, the result of recommendation is not stored and he turns back to his main page. If he clicks on “Confirmar la selección”, a selected variant will be stored like a SOLUTION for a new case. Finally, the case will be learned by a system (stored in a database), a CBR application will pass to a postcycle(), close the connection and switch off. A doctor will see the next pop-up window [*Figure 31*] and, clicking on “OK”, will be redirected to the list of his/her patients:

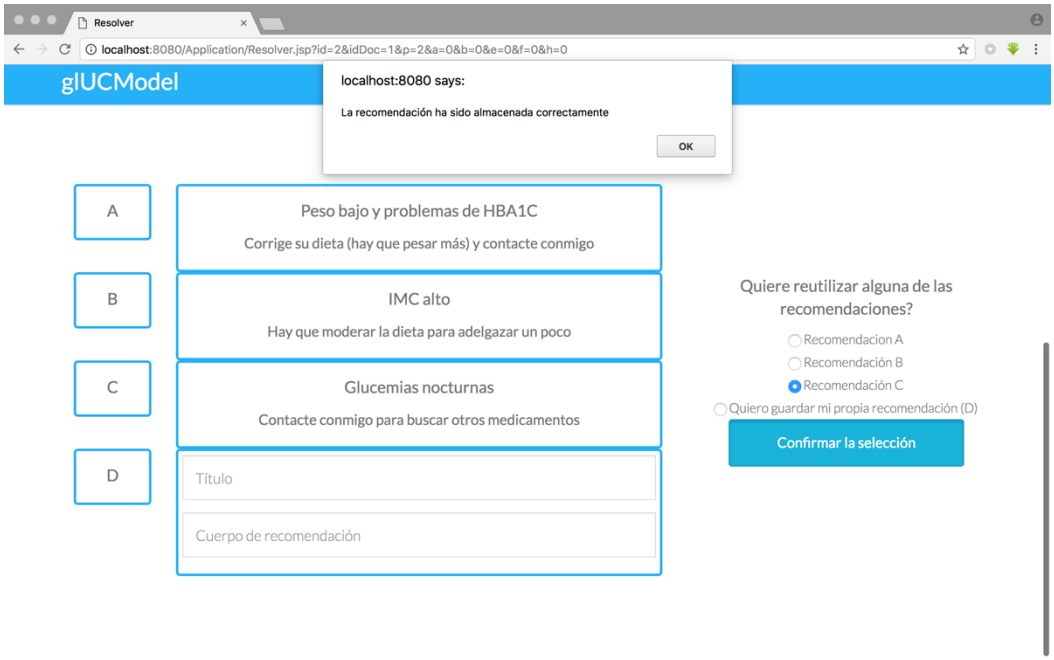

Figure 31: Solution confirmed

Conclusions

Final result

The main result of this project is a creation of a second version of a system “glUCModel 2.0” - and the following objectives were achieved:

- A new database was created: it became smaller, clearer and more focused on a patient (now all records of medical data, messages and investigation cases are linked with concrete persons);
- There is a possibility to download summaries of all 5 types of medical data and upload summaries of 2 of them (weight and glycaemia);
- A new web interface was created. I logically separated menu options from user functions. Furthermore, 3 most important parameters (dynamics of weight and blood sugar level and the level of IMC) are represented with graphics, which makes an application a little bit more intuitive and user-friendly;
- The back-end of application migrated from PHP [35] to Java: now all the operations with data are performed via servlets. A Hibernate technology was also put into practice;
- A recommender was created. Now a doctor is free of comparing a current situation of his patient with hundreds of saved cases: an application will do it instead of him and will purpose the best variants. However, it´s clear that a machine shouldn´t determine a treatment for a human- so; the final decision is always taken by a doctor;

During the development of a system, there were also some professional habits acquired:

- Basic skills of Javascript: this technology was very useful to create dynamic tables with medical data. It was also used to pass data from JSP-pages to servlets and vice versa;
- Basic experience in case-based reasoning systems´ construction: JColibri gives a possibility to create reasonably complex solutions with a

minimum amount of code, which is convenient for those who have never developed something like this;
- Basic skills of Hibernate: this technology helped to manage medical data in easy way and it serves like a bridge between data types defined in Java and in SQL;
- Using of JSTL: JavaServer Pages Standard Tag Library was used to incorporate SQL queries directly into .jsp´s – and this way is much shorter and easier, than performing a request via servlet;
- Initial experience of full-stack development: this project contains both front-end and server-side parts, which had to be configured in appropriate way to let these parts communicate with each other

Resultado final

El resultado final de este proyecto es la creación de la segunda versión de "glUCModel" [3]- y los siguientes objetivos eran cumplidos:

- Fue creada una nueva base de datos: ahora es más pequeña, más clara y más enfocada en el paciente (todos los datos médicos, recomendaciones enviadas y los casos de seguimiento están asociados con unas personas concretas);
- Existe la posibilidad de bajar resúmenes de los 5 tipos de datos existentes y subir resúmenes de 2 tipos (peso y glucemias);
- Fue creada una nueva interfaz Web. He separado las opciones del menú de las funciones del usuario. Además, los 3 parámetros más importantes (dinámicas de peso y glucemias y el nivel de IMC) se representan con gráficos- por tanto, ahora la interfaz es un poco más intuitiva y amigable;
- La parte back-end de la aplicación ha migrado del PHP a Java: ahora todas las manipulaciones con datos se hacen a través de servlets. La tecnología Hibernate también fue incorporada;
- Fue creado el recomendador. Ahora el doctor no tiene que comparar la situación actual de un paciente con centenares de casos almacenados: la aplicación lo hará por él y propondrá las mejores soluciones. Al mismo tiempo, está claro que la máquina no puede determinar el tratamiento para el humano- por tanto, la decisión final siempre la hace el doctor.

Durante el desarrollo del sistema, algunos conocimientos profesionales han sido adquiridos:

- Habilidades básicas de Javascript: esa tecnología se utilizó para crear tablas dinámicas con datos médicos. Además, fue utilizado para pasar datos de páginas JSP a servlets y viceversa;
- Experiencia inicial en construcción de sistemas de razonamiento basados en casos: JColibri da la posibilidad de crear soluciones razonablemente complejas con la cantidad mínima del código, lo que es muy conveniente para los que no han desarrollado nunca este tipo de sistemas;

- Habilidades básicas de Hibernate: esa tecnología ha permitido simplificar la gestión de datos médicos y sirve como un puente entre tipos de datos definidos en Java y en SQL;
- Uso de JSTL: la librería estándar de etiquetas para páginas de JavaServer fue utilizada para incorporar consultas en SQL directamente en los .jsp´s- es una forma más fácil y corta que la realización de una petición a servlet;
- Experiencia de desarrollo completo: este proyecto tiene tanto la parte de usuario, como del servidor- y por tanto fue necesario configurarlo de forma apropiada para que éstas comuniquen correctamente

Future work

As I mentioned above, this project can serve as a base for other developers- and it´s possible to make many improvements, such as:

- **Add other types of data.** Every diabetic person knows, that it´s very important to keep to a special diet- so, it would be great if an application could count a quantity of proteins, lipids and carbohydrates that were consumed by a patient during a day;

- **New ways of data import.** Nowadays all the data can be submitted to an application just on a manual way or from txt files (in this case, records must have a special format). It would be great to implement an algorithm to work with real glucometers and medical scales;

- **Implement a Moodle space.** Many patients, who were diagnosed a diabetes for the first time, have no idea about insulin injections, special food requirements, details of blood analysis, etc. A Moodle can become a place, where doctors and other users will be able to teach persons and to give advices;

Appendix A: fragments of code

Fragment 1: Hibernate configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://www.hibernate.org/dtd/hib
<hibernate-configuration>
  <session-factory>
    <!-- Database connection settings -->
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.password">toor</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/base</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
        <property name="hibernate.current_session_context_class">thread</property>
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.query.factory_class">org.hibernate.hql.classic.ClassicQueryTranslatorFactory</property>

        <mapping resource="beans/Doctores.hbm.xml"/>
        <mapping resource="beans/Ejercicios.hbm.xml"/>
        <mapping resource="beans/Glucemias.hbm.xml"/>
        <mapping resource="beans/Insulinas.hbm.xml"/>
        <mapping resource="beans/Pacientes.hbm.xml"/>
        <mapping resource="beans/Pesos.hbm.xml"/>
        <mapping resource="beans/Pruebas.hbm.xml"/>
        <mapping resource="beans/RecomendacionEnv.hbm.xml"/>
        <mapping resource="beans/Recomendaciones.hbm.xml"/>
        <mapping resource="beans/Seguimiento.hbm.xml"/>

  </session-factory>
</hibernate-configuration>
```

Fragment 2: JavaBean Pesos.java

```
public class Pesos {

    public Pesos() {

    }

    public Pesos(int idPesos, Integer idPaciente, Date fecha, Double valor) {
        this.idPesos = idPesos;
        this.idPaciente = idPaciente;
        this.fecha = fecha;
        this.valor = valor;
    }

    private int idPesos;
    private Integer idPaciente;
    private Date fecha;
    private Double valor;
```

Fragment 3: mapping of JavaBean Pesos.java

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="beans.Pesos" table="Pesos" schema="base">
        <id name="idPesos">
            <column name="idPesos" sql-type="int(11)"/>
        </id>
        <property name="idPaciente">
            <column name="idPaciente" sql-type="int(11)" not-null="true"/>
        </property>
        <property name="fecha">
            <column name="fecha" sql-type="date" not-null="true"/>
        </property>
        <property name="valor">
            <column name="valor" sql-type="double" precision="-1" not-null="true"/>
        </property>
    </class>
</hibernate-mapping>
```

Fragment 4: class HIbernateUtil.java

```java
package database;

import org.hibernate.SessionFactory;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            // Create the SessionFactory from standard (hibernate.cfg.xml)
            // config file.
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed. " + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Fragment 5: delete() operation using Hibernate

```java
  public void deletePes(int pesId) {
      Session session = database.HibernateUtil.getSessionFactory().openSession();
      Transaction tx = session.beginTransaction();
      try {
          Pesos pes = (Pesos) session.load(Pesos.class, new Integer(pesId));
          session.delete(pes);
          tx.commit();
      } catch (RuntimeException e) {
          if (tx != null) {
              tx.rollback();
          }
          e.printStackTrace();
      } finally {
          session.flush();
          session.close();
      }
  }
```

Fragment 6: class CrudPac.java

```
package crud;

import java.util.ArrayList;

public class CrudPac {
    public List<Pacientes> allPac(int prior, int start, int pages) {

    public int getSize(int length) {

    public int getTotalSize() {

    public void addPac(Pacientes pac) {

    public void deletePac(int pacId) {

    public void updatePac(Pacientes pac) {

    public void registerWeight(Pacientes pac) {
}
```

Fragment 7: database configuration for recommender

```
<DataBaseConfiguration>
    <HibernateConfigFile>hibernate.cfg.xml</HibernateConfigFile>
    <DescriptionMappingFile>recom/CaseDescription.hbm.xml</DescriptionMappingFile>
    <DescriptionClassName>recom.CaseDescription</DescriptionClassName>
    <SolutionMappingFile>recom/CaseSolution.hbm.xml</SolutionMappingFile>
    <SolutionClassName>recom.CaseSolution</SolutionClassName>
</DataBaseConfiguration>
```

Fragment 8: servlet ContrPac.java

```
public class ContrPac extends HttpServlet {
    private CrudPac dao = new CrudPac();
    private static final long serialVersionUID = 1L;
    private HashMap<String, Object> JSONROOT = new HashMap<String, Object>();
    private int counter;

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        String action = request.getParameter("action");
        if (action != null) {
            List<Pacientes> pacList = new ArrayList<Pacientes>();

            Gson gson = new GsonBuilder().setPrettyPrinting().create();
            response.setContentType("application/json");

            if (action.equals("list")) {
                try {
                    int startPageIndex = Integer.parseInt(request.getParameter("jtStartIndex"));
                    int recordsPerPage = Integer.parseInt(request.getParameter("jtPageSize"));
                    pacList = dao.allPac(Integer.parseInt(request.getParameter("id")), startPageIndex, recordsPerPage);
                    counter = dao.getSize(Integer.parseInt(request.getParameter("id")));

                    // Return in the format required by jTable plugin
                    JSONROOT.put("Result", "OK");
                    JSONROOT.put("Records", pacList);
                    JSONROOT.put("TotalRecordCount", counter);
```

Fragment 9: declaration of servlets

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
  <display-name>Application</display-name>
  <welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<servlet>
  <servlet-name>ContrEjer</servlet-name>
  <servlet-class>servlet.ContrEjer</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ContrEjer</servlet-name>
  <url-pattern>/ContrEjer</url-pattern>
</servlet-mapping>
```

Fragment 10: use of directives

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
```

Fragment 11: JSTL insertion

```jsp
<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
   url="jdbc:mysql://localhost/base"
   user="root"  password="toor"/>
<sql:query dataSource="${snapshot}" var="result">
   SELECT * from Pacientes where idPacientes = <%= request.getParameter("id") %>;
</sql:query>
```

Fragment 12: configuration of a recommender

```
public void configure() throws ExecutionException {
    try {
        // database.DBServer.init();
        _connector = new jcolibri.connector.DataBaseConnector();
        _connector.initFromXMLfile(jcolibri.util.FileIO.findFile("database/databaseconfig.xml"));
        _caseBase = new LinealCaseBase();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Fragment 13: precycle()

```
public CBRCaseBase preCycle() throws ExecutionException {
     _caseBase.init(_connector);
     Collection<CBRCase> cases = _caseBase.getCases();
     for (CBRCase c : cases)
         System.out.println(c);
     return _caseBase;
}
```

Fragment 14: cycle() and configuration of a "filter"

```
public void cycle(CBRQuery query) throws ExecutionException {
    NNConfig config = new NNConfig();
    config.setDescriptionSimFunction(new Average());

    Attribute at = new Attribute("idPat", CaseDescription.class);
    config.addMapping(at, new Equal());
    config.setWeight(at, 1.5);

    at = new Attribute("h1", CaseDescription.class);
    config.addMapping(at, new Interval(7));
    config.setWeight(at, 1.0);

    at = new Attribute("h2", CaseDescription.class);
    config.addMapping(at, new Interval(7));
    config.setWeight(at, 1.0);
```

Fragment 15: retrieving results

```
// Calculating result
Collection<RetrievalResult> eval = NNScoringMethod.evaluateSimilarity(_caseBase.getCases(), query, config);
Collection<CBRCase> selectedcases = SelectCases.selectTopK(eval, K);
// Asking for a best solution
eval = SelectCases.selectTopKRR(eval, K);
Iterator<RetrievalResult> it = eval.iterator();
while (it.hasNext()) {
    res = (RetrievalResult) it.next();
    desc = (CaseDescription) res.get_case().getDescription();
    sol = (CaseSolution) res.get_case().getSolution();
    // Answer- in a real system it will return just a sol.recs
    System.out.println(desc);
    System.out.println("Con factor " + res.getEval() + " la recomendación correcta es " + sol.getSolutionId()
            + ": " + sol.getSolutionTitle() + ". " + sol.getSolutionDesc());
}
```

Video demonstration of a system

To see a demonstration of the system, please, follow the next link:
https://youtu.be/ja8TJEzutGo

Bibliography

[1] World Health Organisation, "Diabetes Programme," 6 April 2016. [Online]. Available: http://www.who.int/diabetes/en/.

[2] "DACyA Group, UCM," [Online]. Available: https://www.ucm.es/dacya.

[3] J. I. Hidalgo, E. Maqueda, J. L. Risco-Martín, A. Cuesta-Infante, J. M. Colmenar and J. Nobel, "glUCModel: A monitoring and modeling system for chronic diseases applied to diabetes," *Journal of Biomedical Informatics,* p. 10, 2014.

[4] "Case- Based Reasoning," [Online]. Available: https://en.wikipedia.org/wiki/Case-based_reasoning.

[5] "Diabetes mellitus," [Online]. Available: https://en.wikipedia.org/wiki/Diabetes_mellitus.

[6] U. C. d. Madrid, "glUCModel-Android," 2015. [Online]. Available: https://play.google.com/store/apps/details?id=es.ucm.absys.glucmodel.

[7] U. C. d. Madrid, "glUCModel-iOS," 2015. [Online]. Available: https://itunes.apple.com/es/app/glucmodel/id945626329?mt=8.

[8] "HTML," [Online]. Available: https://en.wikipedia.org/wiki/HTML.

[9] "CSS," [Online]. Available: https://en.wikipedia.org/wiki/Cascading_Style_Sheets.

[10] "Bootstrap," [Online]. Available: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework).

[11] "Responsive web design," [Online]. Available: https://en.wikipedia.org/wiki/Responsive_web_design.

[12] "JavaScript," [Online]. Available: https://en.wikipedia.org/wiki/JavaScript.

[13] "Ajax," [Online]. Available: https://en.wikipedia.org/wiki/Ajax_(programming).

[14] "Apache Tomcat," [Online]. Available: https://en.wikipedia.org/wiki/Apache_Tomcat.

[15] "jColibri," [Online]. Available: http://gaia.fdi.ucm.es/research/colibri/jcolibri.

[16] "Java," [Online]. Available: https://en.wikipedia.org/wiki/Java_(programming_language).

[17] "MySQL," [Online]. Available: https://en.wikipedia.org/wiki/MySQL.

[18] "Hibernate," [Online]. Available: https://en.wikipedia.org/wiki/Hibernate_(framework).

[19] "XML," [Online]. Available: https://en.wikipedia.org/wiki/XML.

[20] "Java annotation," [Online]. Available: https://en.wikipedia.org/wiki/Java_annotation.

[21] J. L. Kolodner, "An introduction to case-based reasoning," *Artificial Intelligence Review,* vol. 6, no. 1, 1992.

[22] C. Marling, M. Sqalli, E. Rissland, H. Muñoz-Avila and D. Aha, "Case-Based Reasoning Integrations," *AI Magazine,* vol. 23, no. 1, 2002.

[23] R. Schank, Dynamic Memory: A Theory of Learning in Computers and People, New York: Cambridge University Press, 1982.

[24] J. L. Kolodner, "Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model," 1982.

[25] M. Lebowitz, "Memory-Based Parsing," *Artificial Intelligence,* vol. 21, pp. 363-404, 1983.

[26] S. Begum, M. Uddin Ahmed, P. Funk, N. Xiong and M. Folke, "Case-Based Reasoning Systems in the Health Sciences: A Survey of Recent Trends and Developments," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 41, no. 4, pp. 421-434, July 2011.

[27] A. Aamodt and E. Plaza, "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," *AI Communications,* vol. 7, no. 1, pp. 39-59, 1994.

[28] B. a. J. A. Smyth, "The adaptive Web. Chapter 20: Recommendations to groups," 2007. [Online]. Available: http://dfki.de/~jameson/pdf/adaptive-web.jameson.pdf.

[29] B. Agudo Calvo, J. del Valle Corral and J. L. Jorro Aragoneses, "Sistema de Recomendación de Actividades Turísticas: Madrid Live".

[30] "Glycated_hemoglobin," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Glycated_hemoglobin.

[31] "JSP Standard Tag Library," [Online]. Available: https://jstl.java.net/.

[32] "Moodle - Open Source Software for Online Learning," Moodle Pty Ltd, 2017. [Online]. Available: https://moodle.com/.

[33] "Session - (Hibernate JavaDocs)," [Online]. Available: https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Session.html.

[34] "JTable- a JQuery plugin to create AJAX based CRUD tables," [Online]. Available: http://jtable.org/.

[35] "PHP," [Online]. Available: https://en.wikipedia.org/wiki/PHP.